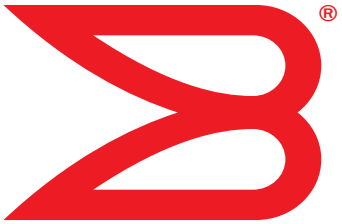
53-1002288-02
August 2011



ServerIron ADX

NAT64 Configuration Guide

Supporting Brocade ServerIron ADX version 12.3.01a

BROCADE

© 2011 Brocade Communications Systems, Inc. All Rights Reserved.

Brocade, the B-wing symbol, BigIron, DCFM, DCX, Fabric OS, FastIron, IronView, NetIron, SAN Health, ServerIron, TurboIron, and Wingspan are registered trademarks, and Brocade Assurance, Brocade NET Health, Brocade One, Extraordinary Networks, MyBrocade, VCS, and VDX are trademarks of Brocade Communications Systems, Inc., in the United States and/or in other countries. Other brands, products, or service names mentioned are or may be trademarks or service marks of their respective owners.

Notice: This document is for informational purposes only and does not set forth any warranty, expressed or implied, concerning any equipment, equipment feature, or service offered or to be offered by Brocade. Brocade reserves the right to make changes to this document at any time, without notice, and assumes no responsibility for its use. This informational document describes features that may not be currently available. Contact a Brocade sales office for information on feature and product availability. Export of technical data contained in this document may require an export license from the United States government.

Brocade Communications Systems, Incorporated

Corporate and Latin American Headquarters
Brocade Communications Systems, Inc.
130 Holger Way
San Jose, CA 95134
Tel: 1-408-333-8000
Fax: 1-408-333-8101
E-mail: info@brocade.com

Asia-Pacific Headquarters
Brocade Communications Systems China HK, Ltd.
No. 1 Guanghua Road
Chao Yang District
Units 2718 and 2818
Beijing 100020, China
Tel: +8610 6588 8888
Fax: +8610 6588 9999
E-mail: china-info@brocade.com

European Headquarters
Brocade Communications Switzerland Sàrl
Centre Swissair
Tour B - 4ème étage
29, Route de l'Aéroport
Case Postale 105
CH-1215 Genève 15
Switzerland
Tel: +41 22 799 5640
Fax: +41 22 799 5641
E-mail: emea-info@brocade.com

Asia-Pacific Headquarters
Brocade Communications Systems Co., Ltd. (Shenzhen WFOE)
Citic Plaza
No. 233 Tian He Road North
Unit 1308 – 13th Floor
Guangzhou, China
Tel: +8620 3891 2000
Fax: +8620 3891 2111
E-mail: china-info@brocade.com

Document History

Title	Publication number	Summary of changes	Date
ServerIron ADX NAT64 Configuration Guide	*53-1002288-01*	New document	May 2011
ServerIron ADX NAT64 Configuration Guide (	*53-1002288-02*	Documentation for “NAT64 Connection logging” feature added to “IPv6-only client to IPv4 resource” chapter to support Release 12.3.01a)	August 2011

Contents

About This Document

Audience vii
Supported hardware and software vii
Document conventions vii
Text formatting vii
Notes, cautions, and danger notices viii
Notice to the reader viii
Related publications ix
Getting technical help or reporting errors ix
Web access ix
E-mail and telephone access ix

Chapter 1 NAT 64 Overview

Implementation Details 3
Requirements for all NAT64 configurations 3
IPv6-only client to IPv4 components 3
IPv4-only client to IPv6 components 3
Protocol support 4
NAT64 ICMPv6 to ICMPv4 message handling 4
NAT64 full size packet handling 5
NAT64 fragmentation support 5
When installing NAT64 on a ServerIron ADX with a previous config 6
NAT64 Connection logging 6
Configuring NAT64 Connection logging 7
Example of NAT64 Connection logging 7

Chapter 2 IPv6-only client to IPv4 resource

Overview 9
Operation of NAT64 for IPv6-only client to IPv4 resource 9
Configuring NAT64 for IPv6-only client to IPv4 resource 11
Configuring a NAT64 IPv6 prefix 11
Configuring an IPv4 NAT address pool 11
Enabling a ServerIron ADX to delete sticky sessions 12
Disabling sticky behavior 12
Enabling connection logging 12
IPv6-only client to IPv4 resource sample configuration 13

Route injection NAT64 13
NAT 64 route injection example 14
HTTP client IP address insertion 15
Enabling HTTP client IP address insertion 16
Configuring Packet fragmentation with IPv6-only client to IPv4 resource 16
High availability for NAT64 17
Configuring the port pool range for HA 18
Enabling inject active only for HA 18
Displaying NAT64 information 18
Displaying session information 19
Displaying NAT 64 translations 19
Displaying NAT 64 statistics 20
Displaying NAT 64 resources 23
Clearing NAT64 information 25
Clearing NAT64 session entries from the session table 25
Clearing stateless and stateful NAT64 statistics 25
NAT64 Connection logging 26
Configuring NAT64 Connection logging 26
Example of NAT64 Connection logging 27

Chapter 3 IPv4-only client to IPv6 resource

Overview 29
Populating the NAT46 mapping table 29
Operation of NAT46 for IPv4-only client to IPv6 resource 30
Configuring NAT64 for IPv4-only client to IPv6 resource 32
Configuring a NAT64 IPv4 prefix 32
Configuring a NAT64 IPv6 prefix 32
Configuring static mapping 33
Configuring DNS dynamic learning 33
Configuring a back-off interval for DNS discoveries 33
IPv4-only client to IPv6 resource sample configuration 34
Route injection NAT46 35
NAT46 Route injection example 36
Configuring Packet fragmentation with IPv4-only client to IPv6 resource 37
High availability for NAT46 38
Displaying NAT46 information 38
Displaying NAT64 IPv4 to IPv6 address mapping 38
Displaying in-progress NAT64 DNS dynamic learning 39
Clearing NAT46 information 39
Clearing IPv6-IPv4 mappings learned through DNS 40

Chapter 4 Access Control List

How ServerIron processes ACLs 41
Prior to release 12.3.01 41
Beginning with release 12.3.01 and later 41
Rule-based ACLs 42
How fragmented packets are processed 42

Default ACL action 43

Types of IP ACLs 44

ACL IDs and entries 44

ACL entries and the Layer 4 CAM 45
Aging out of entries in the Layer 4 CAM 45
Displaying the number of Layer 4 CAM entries 45
Specifying the maximum number of CAM entries for rule-based ACLs 45

Configuring numbered and named ACLs 46
Configuring standard numbered ACLs 46
Configuring extended numbered ACLs 48
Extended ACL syntax 50
Configuring standard or extended named ACLs 54
Displaying ACL definitions 55
Displaying ACLs using keywords 56

Modifying ACLs 58
Adding, inserting, replacing, or deleting a comment 60

Displaying a list of ACL entries 63

Applying an ACLs to interfaces 64
Reapplying modified ACLs 64

ACL logging 65
Displaying ACL log entries 66
Displaying ACL statistics for flow-based ACLs 67
Clearing flow-based ACL statistics 67

Dropping all fragments that exactly match a flow-based ACL 67
Clearing the ACL statistics 68

Enabling ACL filtering of fragmented packets 68
Filtering fragmented packets for rule-based ACLs 68

Enabling hardware filtering for packets denied by flow-based ACLs 70

Enabling strict TCP or UDP mode for flow-based ACLs 71
Enabling strict TCP mode 71
Enabling strict UDP mode 72
Configuring ACL packet and flow counters 73

ACLs and ICMP 74
Using flow-based ACLs to filter ICMP packets based on the IP packet length 74
ICMP filtering with flow-based ACLs 74

Using ACLs and NAT on the same interface (flow-based ACLs) 77

Displaying ACL bindings 78
Troubleshooting rule-based ACLs 78

Chapter 5 IPv6 Access Control Lists

ACL overview 81
Configuration Notes 82
Processing of IPv6 ACLs 82
Configuring an IPv6 ACL 83
Applying an IPv6 ACL to an interface 90
Displaying ACLs 91
Displaying ACLs bound to an interface 91
Using an ACL to Restrict SSH Access 91
Using an ACL to Restrict Telnet Access 92
Logging IPv6 ACLs 92

Chapter 6 Network Address Translation

Introduction 93
Configuring NAT 93
Configuring static NAT 94
Configuring dynamic NAT 94
NAT configuration examples 95
PAT 99
Forwarding packets without NAT translation 100
Translation timeouts 100
Configuring the NAT translation aging timer 100
Stateless static IP NAT 101
Redundancy 101
Enabling IP NAT 102
Enabling static NAT redundancy 102
Enabling dynamic NAT redundancy 102
Displaying NAT information 103
Displaying NAT statistics 104
Displaying NAT translation 106
Displaying NAT redundancy information 107
Displaying VRRPE information 108
Clearing NAT entries from the table 108

About This Document

Audience

This document is designed for system administrators with a working knowledge of Layer 2 and Layer 3 switching and routing.

If you are using a Brocade Layer 3 Switch, you should be familiar with the following protocols if applicable to your network – IP, RIP, OSPF, BGP, ISIS, IGMP, PIM, DVMRP, and VRRP.

Supported hardware and software

Although many different software and hardware configurations are tested and supported by Brocade Communications Systems, Inc. for 12.3.00 documenting all possible configurations and scenarios is beyond the scope of this document.

The following hardware platforms are supported by this release of this guide:

- ServerIron ADX 1000
- ServerIron ADX 4000
- ServerIron ADX 8000
- ServerIron ADX 10K

NOTE
The NAT64 gateway can't be combined to other ServerIron ADX features such as SLB, GSLB, TCS, FWLB etc. This guide includes all features that can be enabled with a NAT64 gateway.

Document conventions

This section describes text formatting conventions and important notice formats used in this document.

Text formatting

The narrative-text formatting conventions that are used are as follows:

bold text	Identifies command names
	Identifies the names of user-manipulated GUI elements
	Identifies keywords
	Identifies text to enter at the GUI or CLI
italic text	Provides emphasis
	Identifies variables
	Identifies document titles
code text	Identifies CLI output

For readability, command names in the narrative portions of this guide are presented in bold: for example, **show version**.

Notes, cautions, and danger notices

The following notices and statements are used in this manual. They are listed below in order of increasing severity of potential hazards.

NOTE
A note provides a tip, guidance or advice, emphasizes important information, or provides a reference to related information.

CAUTION

A Caution statement alerts you to situations that can be potentially hazardous to you or cause damage to hardware, firmware, software, or data.

DANGER

A Danger statement indicates conditions or situations that can be potentially lethal or extremely hazardous to you. Safety labels are also attached directly to products to warn of these conditions or situations.

Notice to the reader

This document may contain references to the trademarks of the following corporations. These trademarks are the properties of their respective companies and corporations.

These references are made for informational purposes only.

Corporation	Referenced Trademarks and Products
Sun Microsystems	Solaris

Corporation	Referenced Trademarks and Products
Microsoft Corporation	Windows NT, Windows 2000
The Open Group	Linux

Related publications

The following Brocade documents supplement the information in this guide:

- *Release Notes for ServerIron Switch and Router Software TrafficWorks 12.2.00*
- *ServerIron ADX Graphical User Interface*
- *ServerIron ADX Server Load Balancing Guide*
- *ServerIron ADX Advanced Server Load Balancing Guide*
- *ServerIron ADX Global Server Load Balancing Guide*
- *ServerIron ADX Security Guide*
- *ServerIron ADX Administration Guide*
- *ServerIron ADX Switch and Router Guide*
- *ServerIron ADX Firewall Load Balancing Guide*
- *ServerIron ADX Hardware Installation Guide*
- *IronWare MIB Reference*

Getting technical help or reporting errors

Brocade is committed to ensuring that your investment in our products remains cost-effective. If you need assistance, or find errors in the manuals, contact Brocade using one of the following options:

Web access

The Knowledge Portal (KP) contains the latest version of this guide and other user guides for the product. You can also report errors on the KP.

Log in to my.Brocade.com, click the **Product Documentation** tab, then click on the link to the Knowledge Portal (KP). Then click on **Cases** > **Create a New Ticket** to report an error. Make sure you specify the document title in the ticket description.

E-mail and telephone access

Go to http://www.brocade.com/services-support/index.page for the latest e-mail and telephone contact information.

Chapter

NAT 64 Overview

1

When the Internet Assigned Numbers Authority (IANA) standardized IPv4 in 1981, no one could have foreseen that its seemingly plentiful pool of 4 billion addresses would become depleted. But according to Internet World Stats, Internet usage grew by 444.8 percent between 2000 and 2010.1 With the increasing deployment of Internet-enabled mobile devices, smart-grid devices, and cloud-based applications, the spike in usage is only going to increase in this decade.

Indeed, on February 3, 2011, the IANA allocated its last block of IPv4 addresses to five Regional Internet Registries (RIRs) around the world. These registries assign IP addresses to Internet Service Providers (ISPs), which in turn issue IP addresses for home and office machines, smart phones, and other Internet-enabled devices.

Taking the place of the old protocol, IPv6 is designed not only to solve IPv4's address scaling challenge but also to rectify other shortcomings. Because of these new capabilities, however, the designers of IPv6 were not able to make it backward-compatible with IPv4. This means devices speaking different versions of Internet Protocol can no longer communicate with each other natively, and applications that rely on such communication will fail.

The industry faces a challenging transition while it moves carefully from its current IPv4-capable routers, switches, servers, and applications to IPv6-ready devices. Service providers—whether they are providing content, hosting services, or Internet access—cannot add or accommodate new customers unless their content is equally accessible to both IPv4 and IPv6 users.

Similarly, e-commerce sites need to accommodate customers without knowing which protocol those customers' client devices use. At the same time, multiple federal governments around the world have enacted regulations forcing their agencies to adopt IPv6. As a result, service providers, hosting services, and other content providers need to provide transition approaches during the evolution period. Today there are several technologies that create a bridge between IPv4 and IPv6; they use such techniques as translation, coexistence, tunneling, overlay, and more. The Network (and Port) Address Translation between IPv6 to IPv4 model—generally known as NAT64—is a mechanism for both the transition to IPv6 and the coexistence of IPv4 with IPv6.

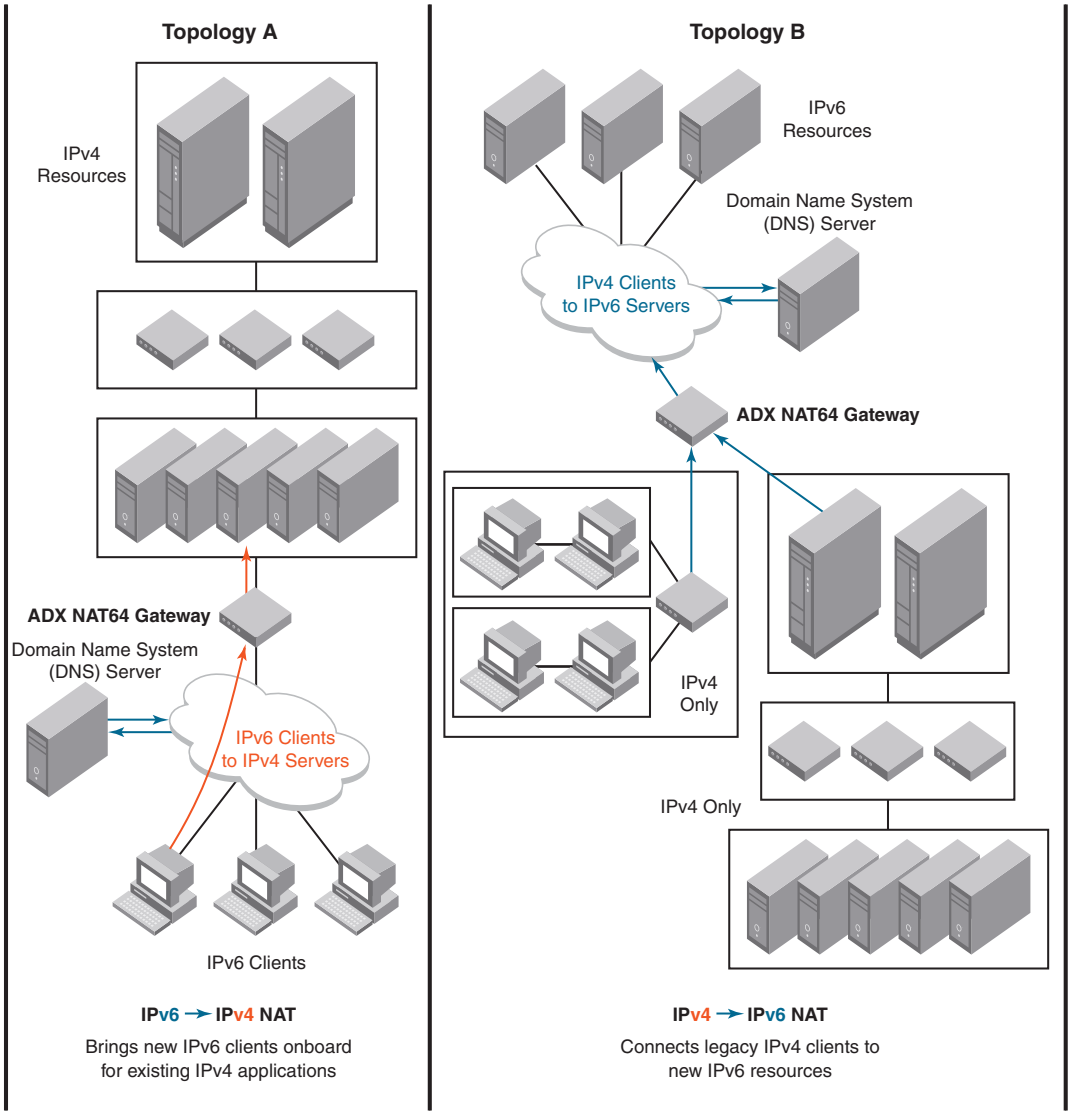
NAT64 works with DNS64, essentially a DNS translation service, to enable client-server communication between an IPv6-only client and an IPv4-only server and vice versa. It allows for peer-to-peer communication where communication can originate from an end-node running either of the two protocols. NAT64 utilizes a preassigned IPv6 prefix to algorithmically translate IPv4 addresses of IPv4 servers. Similarly, it translates the IPv6 addresses of IPv6 servers to and from IPv4 addresses by installing mappings. Overall, the NAT64 model offers a non-intrusive and seamless transition path for organizations looking to explore IPv6.

To facilitate seamless communication with the new breed of IPv6-only customers in addition to IPv4 customers, the Brocade ServerIron ADX Series offers a simple and cost-effective transition path to IPv6 using a standards-based NAT64 gateway.

Topology A: The NAT64 gateway capabilities of Brocade ServerIron ADX enables organizations to bring new IPv6 customers onboard while utilizing their existing IPv4-based infrastructures.

Topology B: Similarly, Brocade ServerIron ADX enables organizations to offer new IPv6-based services to their existing IPv4 clients.

FIGURE 1 NAT64 topologies'



Advantages of NAT64

- It is completely transparent to end-users because address translation occurs at the service provider network edge and it involves no change in client-end CPE devices. Thus it can be an extremely cost-effective and practical solution.
- It allows for transition to IPv6 while preserving existing IPv4-based infrastructure. It facilitates coexistence of IPv4-only and IPv6-only devices while ensuring seamless communication between the two during the transition period.

Disadvantages of NAT64

- It involves translating addresses between IPv4 and IPv6, resulting in potential loss of originating client IP addresses unless they are captured through some other means such as client ip insertion available on the ServerIron ADX.

Implementation Details

This section describes the components required for a NAT64 implementation as well as information that is common to the IPv6-only client to IPv4 resource configuration as described in Chapter 2, “IPv6-only client to IPv4 resource”and the IPv4-only client to IPv6 resource configuration as described in Chapter 3, “IPv4-only client to IPv6 resource”.

NOTE
The NAT64 gateway can't be combined to other ServerIron ADX features such as SLB, GSLB, TCS, FWLB etc. This guide includes all features that can be enabled with a NAT64 gateway.

Requirements for all NAT64 configurations

The following are required in a IPv6-only client to IPv4 resource configuration:

- The NAT64 mechanism is implemented on a ServerIron ADX that has (at least) two interfaces: an IPv4 interface connected to the IPv4 network, and an IPv6 interface connected to the IPv6 network. IPv6 must be enabled on the ServerIron ADX.
- The DNS64 resolver provides AAAA responses for any resource that has an IPv4 address but not an IPv6 address. It generates the IPv6 address for the resource by concatenating a NAT64 prefix to its IPv4 address. When an IPv6 client queries DNS for the address of an IPv4-only resource, it receives the IPv6 address generated by the DNS64 resolver. Since you can optionally configure your DNS servers with these IPV6 addresses manually, this component is not required

IPv6-only client to IPv4 components

The following is required in a IPv6-only client to IPv4 resource configuration:

NAT64 gateway

The NAT64 gateway receives the IPv6 packet whose Destination IPv6 address was generated by the DNS64 resolver. It then translates the IPv6 address to an IPv4 address used by the resource. Return IPv4 packet from the IPv4 resource is then mapped back to the destination IPv6 address from the client's request. NAT64 is stateful meaning that the NAT64 device keeps track of all the connections between the IPv6 clients and the IPv4 resource.

IPv4-only client to IPv6 components

The following is required in a IPv6-only client to IPv4 resource configuration:

NAT46 gateway

The NAT46 gateway receives the IPv4 packets whose Destination IPv4 address is mapped to an internal IPv6 resource. It then translates the IPv6 address to an IPv4 address used by the resource. Return packets from the IPv6 resource are then mapped back to the clients IPv4 address. NAT46 is stateless, meaning that the NAT64 device does not keep track of the connections between the IPv4 client and the IPv6 resource.

Protocol support

The Brocade implementation of NAT64 supports the following protocols:

- Generic TCP
- Generic UDP
- HTTP
- HTTPS
- ICMP
- RDP

NAT64 ICMPv6 to ICMPv4 message handling

NAT64 ICMP packet handling allows ICMPv6 messages to be translated to corresponding ICMPv4 messages on the IPv4 side and IMCPv4 messages from the IPv4 side to be translated to corresponding ICMPv6 messages on the IPv6 side. Table 1 and Table 2 describe these translations.

TABLE 1 ICMPv6 to ICMPv4 message translation

ICMPv6 Message Type	ICMPv4 Message Type
Destination Unreachable (Type 1) • no route (code 0) • admin prohibited (code 1) • not neighbor (code 2) • address unreachable (code 3) • port unreachable (code 4)	Destination Unreachable (Type 3) • Host Unreachable (code 1) • admin prohibited (code 10) • route fail (code 5) • host unreachable (code 1) • port unreachable (code 3)
Packet Too Big (Type 2)	Destination Unreachable (Type 3) • fragment needed (code 4)
Time Exceeded (Type 3)	Time exceeded (Type 11) • code remains same from ICMP6
Parameter Problem (Type 4) • next header • any other param prob	Type - Dest Unreachable, code - protocol unreachable Type - param prob, code - 0
Echo request (Type 128)	Echo request (Type 8)
Echo Reply (Type 129)	Echo Reply (Type 0)

TABLE 2 ICMPv4 to ICMPv6 message translation

ICMPv4 Message Type	ICMPv6 Message Type
Destination Unreachable (Type 3) • net unreachable (code 0) • host unreachable (code 1) • protocol unreachable (code 2) • port unreachable (code 3) • fragment needed (code 4) • route fail (code 0) • unknown dest network (code 1) • unknown dest host (code 2) • source host isolated (code 3) • dest network admin prohibited (code 4) • dest host admin prohibited (code 0) • network unreachable for tos (code 1) • host unreachable for tos (code 2) • admin prohibit (code 3) • host precedence violated (code 4) • precedence cutoff in effect (code)	Destination Unreachable (Type 1) • no route code (code 0) • no route code(code 0) • type - param prob, code - next header • no port (code 4) • type - packet too big, code- no route • not neighbor code (code 1) • no route code (code 0) • no route code (code 0) • no route code (code 0) • admin prohibited (code 1) • admin prohibited (code 1) • no route code (code 0) • no route code (code 0) • admin prohibited (code 1) • admin prohibited (code 1) • admin prohibited (code 1)
Time Exceeded (Type 3)	Time exceeded (Type 11) • code remains same from ICMP
Parameter Problem (Type 4) • next header • any other param prob	Type - Dest Unreachable, code - protocol unreachable Type - param prob, code - 0
Echo request (Type 8)	Echo request (Type 128
Echo Reply (Type 0)	Echo Reply (Type 129)

NAT64 full size packet handling

NAT64 full size packet handling enables full-sized packets from the IPv4 host to be split into multiple packets for the IPv6 client. This functionality is needed because an IPv4 header is 20 bytes in length and IPv6 header (along with extension headers if applicable) is greater than or equal to 40 bytes. If a ServerIron ADX receives a full-sized packet (total length of IPv4 packet equals to MTU) on IPv4 side, when it translates the IPv4 header to IPv6 header, the total length of the IP packet may exceed the MTU of the IPv6 side, so this full size packet needs to be split into two IPv6 fragments. The head fragment size will be equal to the IPV6 side MTU and the non-head fragment size will be equal to the original packet size minus the first fragment size. After performing this full size packet handling, the ServerIron ADX sends these two fragments to the IPv6 side host.

NAT64 fragmentation support

NAT64 fragmentation handling enables a ServerIron ADX to convert an IPv4 fragmented packet to an IPv6 fragment packet and vice-versa. The fragmentation formats are different for IPv4 and IPv6.

For the IPv6 to IPv4 direction, the ServerIron ADX:

- strips off the IPv6 fragment header
- extracts the information stored in it

- populates the fragment ID, offset, and flags of the IPv4 header

For the IPv4 to IPv6 direction, ADX:

- extracts information stored in the IPv4 header
- creates and populates the IPv6 fragment header

NOTE
Because the nature of the ICMP checksum mechanism in IPv6 is different than in IPv4, ICMP fragmentation is currently not supported, and all fragmented ICMP packets received on either IPv6 or IPv4 will be dropped. There are counters that keep track of the number of packets dropped.

NOTE
When a IPv4 host sends multiple fragments with UDP checksum 0, the translation of those packets from IPv4 to IPv6 is not supported

When installing NAT64 on a ServerIron ADX with a previous config

If a ServerIron ADX was previously configured with any "server ...." configuration statements for SLB, they must be removed prior to creating any NAT64 configuration and the device must be rebooted.

This can be done by either of the following methods:

Option 1: Manually prepend a "no " keyword in front of each complete "server " statement, thus removing each configuration statement or section in a granular fashion. Write the config to memory and then reload.

Option 2: Completely erase the entire configuration with the **erase startup-config** command and then reload without saving.

- - -

NAT64 Connection logging

A ServerIron ADX provides NAT64 connection logging to enable administrators to audit and log NAT64 connections created on the ServerIron ADX. A user can configure the ServerIron ADX to send a message to an external Syslog server each time NAT64 creates session table entries for NAT64 traffic.

The forward flow for NAT64 is from the IPv6 Client to the NAT64 IPv6 prefix::ipv4 destination address. The ServerIron ADX selects a NAT pool IP and port to replace the Client IP and strips off the NAT64 prefix to create the IPv4 destination address.

The logging displays the addresses of the following:

- Protocol
- Client IP
- Client Port
- NAT64 prefix
- IPv4 destination IP
- Destination port

- NAT pool IP
- NAT port

A user can recreate an IPv6 destination IP address by concatenating the NAT64 prefix+ the IPv4 destination IP address.

Currently a Syslog message is sent each time a flow session pair is created. There is no buffering or batching in the current release.

Beginning with release 12.301a, the connection creation is logged. The ServerIron ADX does not currently log connection teardown.

NOTE
This feature is only applicable to Stateful NAT64 since no sessions are created for Stateless NAT64 traffic.

NOTE
Enabling NAT64 logging will have an impact on performance.

Configuring NAT64 Connection logging

To enable NAT64 Connection logging on a ServerIron ADX, you must configure the IP address of the external Syslog Server and enable NAT64 connection-logging.

Configuring the IP address of the external Syslog Server

You can configure the IP address of the external Syslog Server using the following commands.

```
ServerIronADX#configure terminal
ServerIronADX(config)# logging 100.100.100.1
ServerIronADX(config)#
```

Syntax: [no] logging <*ip_address*>

Enabling NAT64 connection-logging

You can enable NAT64 connection-logging using the following commands.

```
adx-nat64#conf t
adx-nat64(config)#nat64 connection-log
adx-nat64(config)#
```

Syntax: [no] nat64 connection-log

Example of NAT64 Connection logging

The following example displays Syslog output for NAT64 Connection logging.

```
USER.INFO Jul 13 02:44:47 192.168.13.1 NAT64-EST proto=UDP
sip=2013::20c:29ff:fe06:4473 sp=53947 prefix=3013:: dip=192.168.130.200 dp=00053
nip=192.168.130.10 np=37888
USER.INFO Jul 13 02:44:57 192.168.13.1 NAT64-EST proto=TCP
sip=2013::20c:29ff:fe06:4473 sp=35659 prefix=3013:: dip=192.168.130.200 dp=00053
nip=192.168.130.10 np=37889
```

TABLE 3 Display fields for NAT64 connection logging

This field...	Displays...
USER.INFO	Informational message
Time stamp	The date and time the message was logged.
NAT64-EST	NAT64 session creation event.
proto	The protocol (UDP or TCP)
sip	The client IP address.
sp	The client port ID.
prefix	The NAT64 prefix in the destination address.
dip	The IPv4 address in the destination IPv6 address.
dp	The destination port ID.
nip	The NAT64 NAT pool IP address.
np	The NAT64 NAT port ID.

Chapter*

IPv6-only client to IPv4 resource

2

Overview

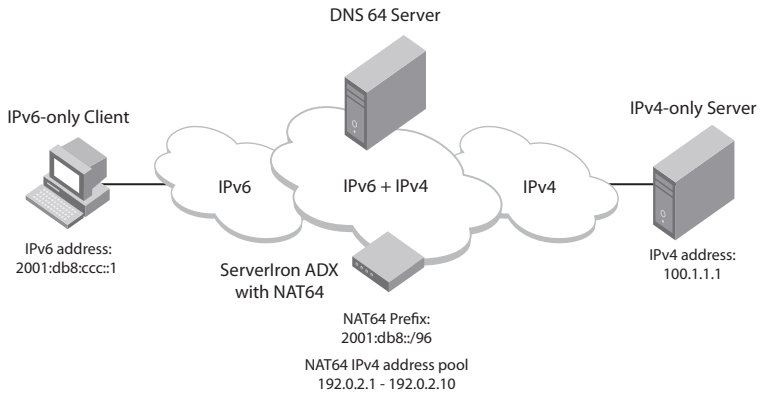
A NAT64 gateway enables IPv6-only clients and IPv4 resources to communicate with each other via address and packet translation. This translation operation is performed on the NAT64 gateway using stateful sessions. This mode requires the following components:

- NAT64 Prefix – The NAT64 prefix converts the IPv4 address of the IPv4-only resource to an IPv6 address that the IPv6 client can send a request to. It is created by defining an IPv6 prefix that is then concatenated to the beginning of the IPv4 address of the IPv4 resource.
- DNS server – This component (not supplied by Brocade) is responsible for providing the IPv6 client with the IPv6 address (defined using the NAT64 prefix required) to reach the IPv4 resource.
- NAT64 gateway – This component is installed on a ServerIron ADX. It is reached by the IPv6 client using the IPv6 address supplied to it by the DNS server. When packets from the IPv6 server reach the NAT64 gateway a stateful session is created during which the IPv6 destination address sent by the client to an IPv4 resource has its IPv6 prefix stripped and the packet is forwarded to the IPv4-only resource. Because the state is maintained, packets returned from the IPv4-only resource to the IPv6-only client are able to use the same session to perform the reverse translation required.

Operation of NAT64 for IPv6-only client to IPv4 resource

Figure 2 provides a high-level view of the IPv6-only client to IPv4 resource NAT64 configuration. As shown, the client resides in an IPv6 only network and the server (resource) resides in an IPv4 only network. The DNS64 server and the ServerIron ADX (configured as a NAT64 gateway) communicate with both the IPv4 and IPv6 networks.

FIGURE 2 IPv6-only client to IPv4 resource overview



The DNS64 server (not supplied by Brocade) is configured to respond to a query from the IPv6 client with an IPv6 address created from the NAT64 prefix and the IPv4 address of the IPv4-only server. In the example shown in Figure 3, the IPv6 client sends a query to the DNS64 server for the IP address of “www.brocadetest.com” and the DNS server responds with the IPv6 address “2001:db8::100.1.1.1”. Notice that this address is made up of the NAT64 prefix (2001:db8::) and the IP address of the IPv4-only server (100.1.1.1).

FIGURE 3 IPv6 client to DNS server communication

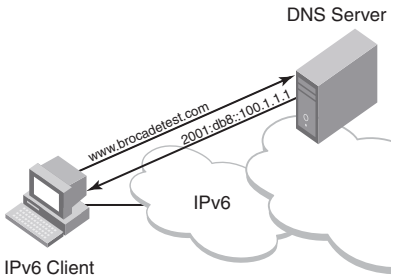
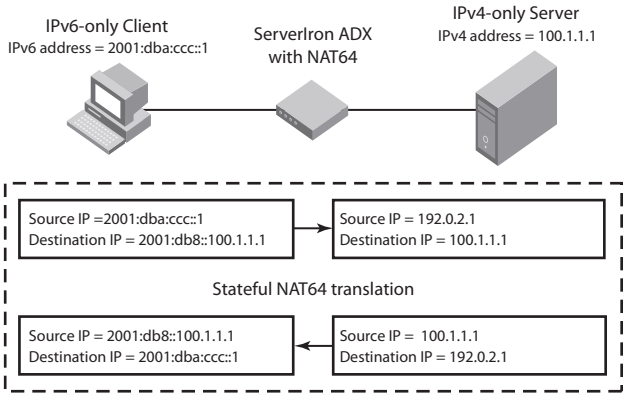


Figure 4 illustrates the stateful NAT64 translation. In this example, the packet is sent with the IPv6 source address of the client (2001:dba:ccc::1) to the destination IPv6 address that was obtained for “brocadetest.com” (2001:db8::100.1.1.1) from the DNS64 server. The NAT64 gateway then selects the IP address “192.0.2.1” from the assigned pool (192.0.2.1 - 192.0.2.10) as its source IPv4 address. It strips the IPv6 prefix from the destination IPv6 address that was sent from the IPv6-only client and sets the remaining IPv4 address (100.1.1.1) as the new destination address. Because NAT64 is stateful, the NAT64 gateway keeps track of all its connections. When a return packet from the IPv4 server destined for the IPv6-only client arrives at the NAT64 gateway, it is able to map the packet with the destination address “192.0.2.1” (from the pool) to the client’s IPv6 address (2001:dba:ccc::11), and the IPv4-only server’s source address (100.1.1.1) back to the IPv6 prefixed address.

FIGURE 4 Stateful NAT64 translation



NOTE
If the ServerIron ADX receives an IPv6 packet that contains a protocol other than TCP, UDP, or ICMPv6 in the last Next Header, then the packet should be discarded silently

Configuring NAT64 for IPv6-only client to IPv4 resource

Basic configuration of NAT64 for an IPv6-only client to an IPv4-only resource is performed using the following:

- Configure a NAT64 IPv6 prefix (required)
- Configure an IPv4 NAT address pool (required)
- Enable the ServerIron ADX to delete sticky sessions (optional)
- Enable connection logging (optional)

Other configuration options are described later in the chapter.

NOTE
If you are configuring NAT64 on a system that has had a previous SLB configuration see "When installing NAT64 on a ServerIron ADX with a previous config" on page 6

Configuring a NAT64 IPv6 prefix

A NAT64 IPv6 prefix is configured on a ServerIron ADX as shown.

```
ServerIron ADX(config) nat64 ipv6-prefix 2001:db7:8000::/96
```

Syntax: **[no] nat64 ipv6-prefix** <prefix/subnet> [**inject-static-route** { **ve** <port-number> | **ethernet** <port-number> }]

NOTE
The maximum number of NAT64 prefixes that can be configured is 8.

The <prefix/subnet> variable specifies the NAT64 IPv6 prefix that will be used by the ServerIron ADX when operating as a NAT64 gateway.

The **inject-static-route** option is used to advertise the subnet defined by the <prefix/subnet> variable to the IPv6 network. It is configured by an IPv6 interface specified by the **ve** <port-number> or **ethernet** <port-number> variable. It should be the interface connected to the adjacent router. This option is described in detail in "Route injection NAT64" on page 13.

Configuring an IPv4 NAT address pool

You can configure a NAT64 IPv4 NAT address pool as shown.

```
ServerIron ADX(config) nat64 pool nat64-zone1 201.1.1 201.1.1.20 prefix-length 24
```

Syntax: [**no**] **nat64 pool** <*pool-name*> <*start-IPaddress*> <*end-IPaddress*> **prefix length** <*prefix length*> [**port-pool-range** { **1** | **2** }] [**inject-static-route**]

NOTE
The maximum number of IP addresses that can be configured in a NAT pool is 192.

The <*pool-name*> variable specifies the name of the NAT64 IPv4 pool. The name can be up to 255 characters in length.

The <*start-IPaddress*> variable specifies the IPv4 address at the beginning of the NAT64 pool range. This value should be the lowest numbered IPv4 address in the range.

The <*end-IPaddress*> variable specifies the IPv4 address at the end of the NAT64 pool range. This value should be the highest numbered IPv4 address in the range.

The **prefix length** option uses the CIDR prefix value specified by the <*prefix-length*> variable to distinguish the portion of the IPv4 address that will be used for all IPv4 addresses in the pool. For example, you can use “24” for the length of the CIDR prefix.

The **port-pool-range** option enables NAT64 redundancy. A value of **2** specifies a higher priority of the NAT IP than a value of **1**. A value of **2** also indicates that source ports allocated for the NAT IP are from the higher range.

The **inject-static-route** option injects the <*start-IPaddress*> /<*prefix-length*> subnet route into routing table which is then advertised to the adjacent router. This option is described in detail in “Route injection NAT64” on page 13.

NOTE
If the ServerIron ADX runs out of Stateful NAT64 NAT pool ports or session entries, then a new connection request should be dropped silently.

Enabling a ServerIron ADX to delete sticky sessions

You can configure a ServerIron ADX to delete a sticky session and select a new NAT64 pool IPv4 address if a pool in sticky session mode does not have available ports. This is configured as shown.

```
ServerIron ADX(config) nat64 nat-sticky-delete
```

Syntax: [no] **nat64 nat-sticky-delete**

If the NAT pool IP in a sticky session runs out of available ports, and a new IPv6 client connection request arrives, then with this command the ServerIron ADX deletes the existing sticky session, selects a new NAT pool IP and creates a new sticky session for this IPv6 client/NAT pool IP pair. Any existing sessions from this IPv6 client will continue to exist, however all new connections will use the new sticky session thus created."

Disabling sticky behavior

You can disable sticky behavior and select new NAT pool IP each time for an IPv6 client. This is configured as shown.

```
ServerIron ADX(config) nat64 disable-sticky
```

Syntax: [no] **nat64 disable-sticky**

Enabling connection logging

You can enable connection logging to the syslog as shown.

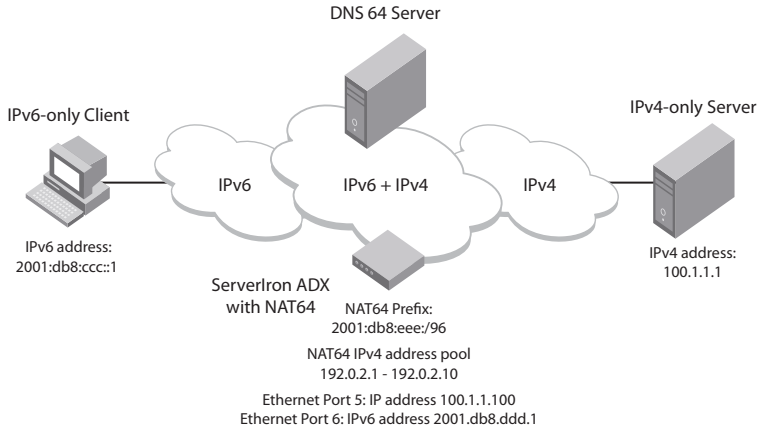
```
ServerIron ADX(config) nat64 connection-log
```

Syntax: [no] **nat64 connection-log**

IPv6-only client to IPv4 resource sample configuration

The following example provides the commands required on a ServerIron ADX to enable the basic IPv6-only client to IPv4 resource configuration shown in Figure 5.

FIGURE 5 NAT64 stateful example




```
ServerIron ADX(config) nat64 pool nat54-zone1 192.0.2.1 192.0.2.10 prefix-length
24
ServerIron ADX(config) nat64 ipv6-prefix 2001:db8:eee::/96
ServerIron ADX(config)interface ethernet 5
ServerIron ADX(config-if-1000-5) ip address 100.1.1.100 255.255.255.0
ServerIron ADX(config-if-1000-5) exit
ServerIron ADX(config)interface ethernet 6
ServerIron ADX(config-if-1000-6) ipv6 address 2001:db8:ddd::1/64
```

Route injection NAT64

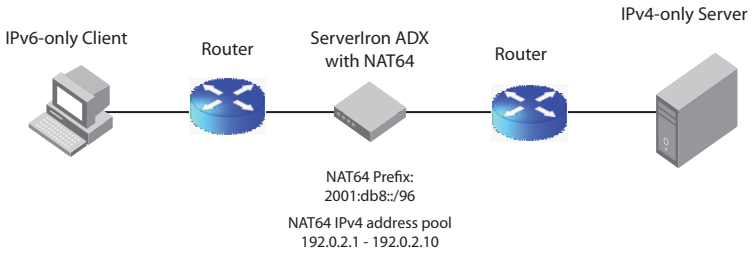
As described previously, IPv6 prefix addresses are defined on the ServerIron ADX to represent IPv4 resources to IPv6 clients. Also, a pool of IPv4 addresses is defined on the ServerIron ADX to represent the IPv6 clients to the IPv4 resources. The addresses assigned for these translations need to be made available as destinations in the routing tables of the IPv6 and IPv4 networks that they reside in. For this reason, we provide command options that direct the ServerIron ADX to inject routes to these addresses. Route distribution is then performed using any of the routing protocols supported by the ServerIron ADX: OSFP, IS-IS and BGP (IPv4 and IPv6).

NAT64 route injection requires that the ServerIron ADX have a routing adjacency relationship with a router for the protocols you want to support route distribution with. For the IPv4 address pool, the **nat64 pool** command has the **inject-static-route** option which injects a route to the subnet into the local routing table. This route is then advertised as the destination for traffic bound for the subnet by the routing protocol that is configured on the ServerIron ADX.

For resources that are mapped to the subnet defined by the IPv6 prefix, the **nat64 ipv6-prefix** command has an option to identify an interface on the ServerIron ADX (Ethernet or VE). The interface defined here must have an IPv6 address and should be the interface that is directly connected to the adjacent router. This subnet defined by the IPv6 prefix is advertised to the adjacent neighbor by the configured routing protocol and made available in routing tables of the IPv6 network. The next hop to the subnet is advertised to the IPv6 network as the adjacent router.

Figure 6 shows a typical IPv6-only client to IPv4 resource topology configured with router adjacency relationships on both the IPv4 and IPv6 sides of the ServerIron ADX. In this configuration, routes defined by the IPv6 prefix and NAT64 pool are advertised to the adjacent routers and distributed to the respective networks using the routing protocol configured.

FIGURE 6 NAT64 Route injection



For more details about how to configure routing protocols on a ServerIron ADX, see the following chapters in the *ServerIron ADX Switch and Router Guide*:

- Configuring OSPF
- Configuring IPv6 Dynamic Routing
- Configuring IS-IS (IPv4)
- Configuring IPv6 IS-IS
- Configuring BGP4 (IPv4
- Configuring BGP4+

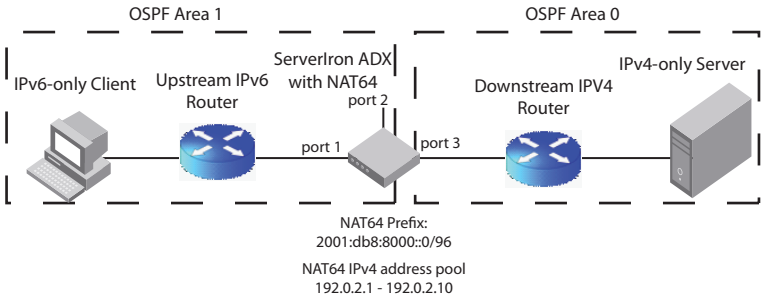
NAT 64 route injection example

The following example configures a ServerIron ADX with NAT64 for route injection into OSPF with the following required details:

- an IPv6 address is configured: VLAN 1 is configured with VE 7 which has an IPv6 address (2001:db8:8000::1)
- OSPF and OSPFv6 are configured for static route redistribution
- The IPv4 side is configured as OSPF Area 0 and the IPv6 side is configured as OSPF Area 1
- The **nat64 pool** command is configured with the **inject-static-route** option.
- The **nat64 ipv6-prefix** command is configured with the **inject-static-route** option which specifies the **ve 7** interface.

For a description of NAT64 route injection see "Route injection NAT64" on page 13.

FIGURE 7 NAT64 route injection example




```
ServerIron ADX(config)# vlan 1
ServerIron ADX(config-vlan-1)# interface ethernet 1 to 2
ServerIron ADX(config-vlan-1)# interface ve 7
ServerIron ADX(config-vlan-1)# exit
ServerIron ADX(config)# interface ve 7
ServerIron ADX(config-vif-7)# ipv6 address 2001:db8:7000::1/64
ServerIron ADX(config-vif-7)# exit
ServerIron ADX(config)# router ospf
ServerIron ADX(config-ospf-router)# redistribute static
ServerIron ADX(config-ospf-router)# area 0
ServerIron ADX(config-ospf-router)# exit
ServerIron ADX(config)# ipv6 router ospf
ServerIron ADX(config-ospf6-router)# redistribute static
ServerIron ADX(config-ospf6-router)# area 1
ServerIron ADX(config-ospf6-router)# exit
ServerIron ADX(config)# nat64 pool nat64-zone1 192.0.2.1 192.0.2.10 prefix-length
24 port-pool-range 1 inject-static-route
ServerIron ADX(config)# nat64 ipv6-prefix 2001:db8:8000::0/96 inject-static-route
ve 7
```

NOTE
While this example describes NAT46 route injection for the OSPF protocol, you can also use this feature with BGP and IS-IS. Only the routing protocol configurations will differ.

HTTP client IP address insertion

When IP addresses are translated, the resource will not be able to get the original client IP address. If client IP address insertion is enabled, the client IP address will be inserted as an HTTP header. The header will be inserted as the last header in the HTTP request.

For example, where the original HTTP request is:

```
GET /abc/index.html HTTP 1/0\r\n
Host: foo.com\r\n
…
Connection: Keep-Alive\r\n
\r\n
```

After insertion, the HTTP request will be:

```
GET /abc/index.html HTTP 1/0\r\n
Host: foo.com\r\n
…
Connection: Keep-Alive\r\n
X-Forwarded-For: 2001:db8::6401:101\r\n
\r\n
```

No automatic detection of HTTP traffic

Client IP address insertion needs to be enabled for the port running HTTP traffic. The ServerIron ADX will not automatically detect HTTP traffic on any port.

Insertion to the first request

A client IP address will only be inserted for the first HTTP request, in a TCP connection. This means that for a keep-alive or pipelined request, the client IP address header is inserted for the first request but not for subsequent requests.

Enabling HTTP client IP address insertion

You can configure HTTP client IP insertion and configure the http header name through the **nat64 http-client-ip-insertion port** command. This is configured as shown.

```
ServerIron ADX(config) nat64 http-client-ip-insertion
```

Syntax: [**no**] **nat64 http-client-ip-insertion** {**port** <port> | **acl** <acl-name>} <header-string>

This command enables you to configure HTTP client IP insertion.

Enabling this command with the **port** option directs client IP insertion for the <port> specified by its decimal value.

Optionally, you can use the acl option to direct client IP insertion as defined by the ACL that is specified by the <acl-name> variable.

If enabled without specifying a <port> or <acl-name>, client IP insertion will be enabled for HTTP port 80 and the header name will be “X-Forwarded-For”.

A customized header can be be added by specifying one for the <header> variable. In The following example, the ServerIron ADX is configured to insert “NAT64-CLIENT-IP” in a request to an IPv4 server as the header instead of the default header of “X-Forwarded-For”.

```
ServerIron ADX(config) nat64 http-client-ip-insertion port 80 “NAT64-CLIENT-IP”
```

Configuring Packet fragmentation with IPv6-only client to IPv4 resource

Reverse packets from the IPv4 server to the IPv6 client can be too large and need to be split into two IPv6 packets. The following describes the criteria for judging that packets are too large:

Regular packets – IP total length greater than 1480 bytes

Fragmented packets – IP total length greater than 1480 + 8 bytes

If the packets exceed these limitations, one of the following actions will be taken:

1. If the **frag-664-reverse-full-sized-pkt** command is configured, the packet will be split and no further actions will be performed.
2. If the condition in step 1 isn't met, and the DF bit is set at the server, the "frag needed" ICMP error message will be sent.
3. If the conditions in steps 1 and 2 aren't met, the packet will be split.

The **frag-664-reverse-full-sized-pkt** command is configured as shown in the following.

```
ServerIronADX(config)# frag-664-reverse-full-sized-pkt
```

Syntax: [no] **frag-664-reverse-full-sized-pkt**

For more information about NAT64 fragmentation support see "NAT64 fragmentation support" on page 5.

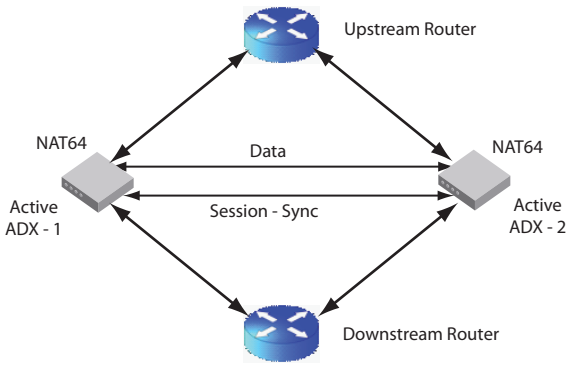
NOTE
If the IPv4 host sends out UDP Fragmented packets with checksum 0, we do not support it.

High availability for NAT64

The only high availability mode currently supported with NAT64 feature is the Active-Active HA. In the Active-Active HA mode, the NAT64 IPv4 NAT pool ports are divided equally among ServerIron ADXs. Upstream and Downstream routers use ECMP to distribute the load equally among the ServerIron ADXs. This ensures that there is no port starvation. This feature is available only for a ServerIron ADX running router code.

Each ServerIron ADX is configured with a NAT64 prefix, specifying the IPv6 address range and a NAT pool that specifies the IPv4 address pool. Sessions are synchronized periodically between the ServerIron ADXs. These sessions are synchronized using a separate link as shown. As long as a ServerIron ADX receives packets on a given flow or session, the age of the session is updated to keep the session alive. The age of the session is also synchronized periodically which maintains the session as alive on both ServerIron ADXs.

FIGURE 8 NAT64 active-active HA configuration



Forwarding can become asymmetric when there are failovers. For example using the topology shown in Figure 8 the following could occur:

1. the link from the downstream router to ADX-1 goes down.
2. ADX-1 receives a request from the upstream router which is forwarded to the downstream router via ADX-2
3. The downstream then forwards the response to the upstream router via ADX-2.
4. The forward and reverse flows hit each of the two ServerIron ADXs which means that the traffic is asymmetric.

Configuring the port pool range for HA

When Active-Active high availability is configured on a ServerIron ADX you can specify the range of source ports allocated for the NAT IP for each of the NAT64 gateways configured within the HA setup, This is enabled by the **port-pool-range** option of the **nat64 pool** command. The **port-pool-range** option enables NAT64 redundancy. A value of **2** specifies a higher priority of the NAT IP then a value of **1**. A value of **2** also indicates that source ports allocated for the NAT IP are from the higher range.

The following example configures a NAT64 IPv4 NAT address pool with the port-pool-range option set to a value of 2.

```
ServerIron ADX(config) nat64 pool nat64-zone1 201.1.1 201.1.1.20 prefix-length 24
port-pool-range 2
```

For details about configuring the **nat64 pool** command, see "Configuring an IPv4 NAT address pool" on page 11.

Enabling inject active only for HA

This command makes sure that only an active ServerIron ADX will inject the static route and the standby ServerIron ADX will withdraw the static routes. Without this command (default configuration) both the active and standby ServerIron ADX will inject the route.

```
ServerIron ADX(config) nat64 inject-active-only
```

Syntax: [**no**] **nat64 inject-active-only**

NOTE
This command only applies to the IPv6-only client to IPv4 resource configuration as described in this chapter because it is performed is the stateful mode and maintains session relationships.

Displaying NAT64 information

You can display the following information for a NAT64 configuration:

- Session information
- NAT64 translations
- NAT64 statistics
- NAT64 resources

Displaying session information

You can use the **show session all** command at the rconsole to display sessions on the ServerIron ADX including NAT64 sessions. NAT64 sessions are indicated by a unique session type in the output. This output is displayed as follows.

```
ServerIron ADX# rconsole 1 1
ServerIron ADX1/1 show session all 0
Session Info:
Flags - 0:UDP, 1:TCP, 2:IP, 3:INT, 4:INVD, H: sessInHash, N: sessInNextEntry
Index Src-IP          Dst-IP            S-port  D-port Age  Server   Flags
===== ======          ======            ======  ======  ===  ======   ========
0     192.168.1.101   200.1.1.2         80      38912   60   n/a      NAT641 H
1     3003::10        0.0.0.0           0       0       57   n/a      NAT641 H
2     3003::10        4003::c0a8:165    1417    80      60   n/a      NAT641 H
```

Syntax: show session all <index>

The <index> variable specifies the integer value of the record in the table that you want to begin the display from. Using the value “0” will display all records in the table.

TABLE 4 Display fields for show session all

This field...	Displays...
Index	The row number of this entry in the Session Table.
Src-IP:	The Source IP address for the session flow.
Dst-IP	The Destination IP address for the session flow.
S-port	The Layer 4 Source Port number for the session flow.
D-port	The Layer 4 Destination Port number for the session flow.
Age	The session age.
Server	The real server name the session belongs to. N/A to NAT64 sessions.
Flags	Flag identifying session type. For example: NAT64 TCP sessions are identified as NAT641. Flags displayed can be: 0:UDP 1:TCP 2:IP 3:INT 4:INVD H: sessInHash N: sessInNextEntry

Displaying NAT 64 translations

You can use the **show nat64 translation** command to display the client IP, NAT IP and Destination IP addresses in a stateful NAT64 gateway. This output is displayed as follows.

```
ServerIron ADX# show nat64 translation
Pro  Client IP          NAT IP             Dest IP
tcp  3003::10|1418      200.1.1.2|38913    192.168.1.101|80
```

Syntax: show nat translation

TABLE 5 Display fields for show nat translation

This field...	Displays...
Pro	The Layer 4 Protocol: TCP. UDP or ICMP.
Client IP:	The IPv6 Client IP address.
NAT IP	The translated IP address from the NAT64 IPv4 pool.
Dest IP	The IP address of the internal IPv4 resource.

Displaying NAT 64 statistics

You can use the **show nat64 statistics** command at the rconsole to display statistics for the NAT64 gateway. This command can be used for stateful and stateless NAT64 configurations. This output is displayed as follows.

```
ServerIron ADX# rconsole 1 1
ServerIron ADX1/1# show nat64 statistics all
*********************
NAT64 Gateway STATISTICS
*********************
         Packet processing:
         Pre-proc internal SIP map create = 18
         Pre-proc create DIP by removing prefix = 17
         Pre-proc pending drop = 0
         Stateless IPv6 prefix prepended = 0
         Stateful IPv6 prefix prepended = 31
         6->4 initiate dynamic learning = 0
         4->6 initiate dynamic learning = 0
         6->4 cannot initiate learning table full = 0
         4->6 cannot initiate learning table full = 0

         IPv6 statistics:
         6->4 pkts recv = 18
         6->4 v6 error = 0

         Fast path statistics:
         6->4 fast path candidate = 18
         6->4 fast path processed pkt = 17
         6->4 fast path processed pkt last sec = 0
         6->4 fast path processed bytes = 2235
         6->4 fast path processed bytes last sec = 0
         6->4 fast path num tcp session lookup = 17
         6->4 fast path num tcp session lookup found = 14
         4->6 fast path candidate = 16
         4->6 fast path processed pkt = 16
         4->6 fast path processed pkt last sec = 0
         4->6 fast path processed bytes = 1371
         4->6 fast path processed bytes last sec = 0
         4->6 nat64 num tcp session lookup = 16
         4->6 fast path nat64 num tcp session lookup found = 0

         Slow path statistics:
         Slow path processed pkt = 0

         Stateless Statistics:
         TCP 6->4 = 0
         TCP 4->6 = 0
```

```
UDP 6->4 = 0
UDP 4->6 = 0
Static pending or error in entry drop = 0

Stateful Statistics:
TCP 6->4 = 17
TCP 4->6 = 16
TCP reverse no session drop = 0
UDP 6->4 = 0
UDP 4->6 = 0
UDP reverse no session drop = 0

NAT64 NAT pool Statistics:
6->4: TCP: NAT port allocated = 3
6->4: TCP: NAT port not available = 0
NAT port freed = 3

NAT64 HA Statistics:
Message sync sent = 0
Message sync received = 0
Error during sending sync messages = 0
Error during receiving sync messages = 0
Invalid prefix index found during message syncing = 0

Errors:
IPv6 Prefix index invalid during pre-pend = 0
Static map entry null in NAT64 proc = 0
Get MTU on V6 outgoing port error = 0

Abnormal packets:
Embedded IPv4 address not unicast = 0
SIP contains IPv6 prefix = 0
```

Syntax: show nat64 statistics [**all** | **stateful** | **stateless** | **dns-dynamic-learning**]

The **all** parameter displays all NAT64 statistics

The **stateful** parameter displays stateful NAT64 statistics only.

The **stateless** parameter displays stateless NAT46 statistics only.

The **dns-dynamic-learning** parameter displays DNS dynamic learning statistics.

TABLE 6 Display fields for show nat64 statistics

This field...	Displays...
Packet processing:	
Pre-proc internal SIP map create =	INTERNAL source IP mappings created.
Pre-proc create DIP by removing prefix =	NAT64 IPv4 destination IPs extracted.
Pre-proc pending drop =	NAT46 (STATELESS) Number of packets dropped due to pending DNS dynamic discovery.
Stateless IPv6 prefix prepended =	Stateless NAT46 V4->V6 packet conversions.
Stateful IPv6 prefix prepended =	Stateful NAT64 v4->v6 packet conversions.
6->4 initiate dynamic learning =	Stateless: Number of DNS dynamic learnings initiated to discover IPv4 address.

TABLE 6 Display fields for show nat64 statistics (Continued)

This field...	Displays...
4->6 initiate dynamic learning =	Stateless: Number of DNS dynamic learnings initiated to discover IPv6 address.
6->4 cannot initiate learning table full =	# DNS learning not initated for for IPv4 address discovery due to table being full.
4->6 cannot initiate learning table full =	# DNS learning not initated for for IPv6 address discovery due to table being full.
IPv6 statistics:	
6->4 pkts recv =	# IPv6 NAT64 packets received.
6->4 v6 error =	NAT64 IPv6 packets received with errors.
Fast path statistics:	
6->4 fast path candidate =	# IPv6 packets identified for optimized IPv4 conversion when nat64 is the only feature enabled.
6->4 fast path processed pkt =	# IPv6 packets that underwent optimized IPv4 conversion when nat64 is the only feature enabled.
6->4 fast path processed pkt last sec =	# of optimized IPv6 to IPv4 converted packets in the last second.
6->4 fast path processed bytes =	# bytes in IPv6 to IPv4 converted packets.
6->4 fast path processed bytes last sec =	# bytes in IPv6 to IPv4 converted packets in the last second.
6->4 fast path num tcp session lookup =	# nat64 TCP session lookups for IPv6 to IPv4 packets.
6->4 fast path num tcp session lookup found =	# successful TCP session lookups for IPv6 to IPv4 packets.
4->6 fast path candidate =	# IPv4 packets identified for optimized IPv6 conversion when nat64 is the only feature enabled
4->6 fast path processed pkt =	# IPv4 packets that underwent optimized IPv6 conversion when nat64 is the only feature enabled.
4->6 fast path processed pkt last sec =	# bytes in IPv4 to IPv6 converted packets in the last second.
4->6 fast path processed bytes =	# bytes in IPv4 to IPv6 converted packets.
4->6 fast path processed bytes last sec =	# bytes in IPv4 to IPv6 converted packets in the last second.
4->6 nat64 num tcp session lookup =	# successful TCP session lookups for IPv4 to IPv6 packets
4->6 fast path nat64 num tcp session lookup found =	# successful TCP session lookups for IPv4 to IPv6 packets.
Slow path statistics:	
Slow path processed pkt = 0	Number of nat64 packets that didn't go through fast-path because of other features being used.
Stateless Statistics:	
TCP 6->4 =	# stateless nat64 TCP IPv6 packets converted to IPv4.
TCP 4->6 =	# stateless TCP nat64 IPv4 packets converted to IPv6.
UDP 6->4 =	# stateless nat64 UDP IPv6 packets converted to IPv4.
UDP 4->6 =	# stateless UDP nat64 IPv4 packets converted toIP v6.
Static pending or error in entry drop =	

TABLE 6 Display fields for show nat64 statistics (Continued)

This field...	Displays...
Stateful Statistics:	
TCP 6->4 =	The number of IPv6 TCP packets that have been translated to IPv4.
TCP 4->6 =	The number of IPv4 TCP packets that have been translated to IPv6.
TCP reverse no session drop =	The number of TCP packets that were dropped due to no reverse session being present.
UDP 6->4 =	The number of IPv6 UDP packets that have been translated to IPv4.
UDP 4->6 =	The number of IPv4 UDP packets that have been translated to IPv6.
UDP reverse no session drop =	The number of UDP packets that were dropped due to no reverse session being present
NAT64 NAT pool Statistics:	
6->4: TCP: NAT port allocated =	The total number of ports allocated from the NAT64 IPv4 pool.
6->4: TCP: NAT port not available =	The total number of occurrences when no NAT64 IPv4 pool port was available.
NAT port freed =	The total number of ports allocated from NAT64 IPv4 pool that have been freed up.
NAT64 HA Statistics:	
Message sync sent =	Number of sessions synced to HA peer.
Message sync received =	Number of NAT64 sessions received from HA peer.
Error during sending sync messages =	# of errors encountered while syncing NAT64 sessions.
Error during receiving sync messages =	# of sync messages with errors (usually indicates config mismatch).
Invalid prefix index found during message syncing =	# of times NAT64 prefix in session received from peer not configured on device.
Errors	
IPv6 Prefix index invalid during pre-pend =	# times when IPv6 prefix wasn't found wehn converting v4->v6 packets.
Static map entry null in NAT64 proc =	Not used.
Get MTU on V6 outgoing port error =	Not used.
Abnormal packets:	
Embedded IPv4 address not unicast =	IPv4 address embedded in the IPv6 prefix is a multi-cast / broadcast address.
SIP contains IPv6 prefix =	Packet received has source ip in NAT64 prefix

Displaying NAT 64 resources

You can use the **show nat64 resources** command at the rconsole to display information about NAT64 resources on theServerIron ADX. This command can be used for stateful and stateless NAT64 configurations. This output is displayed as follows.

```
ServerIron ADX# rconsole 1 1
ServerIron ADX1/1# show nat64 resources
*********************
NAT64 Gateway RESOURCES
*********************
         NAT64 only feature enabled: Yes
         NAT64 Stateless enabled: No
         NAT64 Stateful enabled: Yes

         NAT64 IPv6 prefixes:
         -------------------
         IPv6 Prefix: 4003::
         Number of IPv6 prefixes: 1
         Stateless IPv6 prefix: Not Configured

         NAT64 Stateless:
         -------------------
         IPv6 map hash table size: 1024
         Max mapping entries: 1024
         Number of free map entries: 0


         NAT64 IPv4 NAT Pools:
         -------------------
         Number of NAT pools configured: 1
         Number of NAT pool IPs configured: 1 Maximum: 192
         Head pool: 2591b140
         Current pool: 2591b140
         nat64 pool test:
          start: 200.1.1.1 end: 200.1.1.2 len: 24 port-range: no inject: no
 Pool Mem: 2591b140 Current IP: 200.1.1.2
Ports[0]: Mem: 2591b140 Head: 1 Tail: 0 IP: 200.1.1.1 Memory: 26a2a000 Total: 7168
Free: 7167
Ports[1]: Mem: 2591b140 Head: 0 Tail: 0 IP: 200.1.1.2 Memory: 26a3b000 Total: 7168
Free: 7168
```

Syntax: show nat64 resources

TABLE 7 Display fields for show ip nat statistics

This field...	Displays...
NAT64 Gateway RESOURCES:	
NAT64 only feature enabled:	“Yes” if NAT64 is the only feature configured. “No' otherwise.
NAT64 Stateless enabled:	'Yes” if NAT64 stateless is configured. 'No' otherwise.
NAT64 Stateful enabled: Yes	“Yes” if NAT64 stateful is configured. 'No' otherwise.
NAT64 IPv6 prefixes:	Lists all NAT64 IPv6 prefixes on the system.
IPv6 Prefix:	The first IPv6 prefix on the system. More IPv6 prefixes will be listed similarly.
Number of IPv6 prefixes:	Lists the total number of IPv6 prefixes on the system. max: 8
Stateless IPv6 prefix:	Indicates whether the IPv6 Prefix is configured as stateless.
NAT64 Stateless:	Lists all NAT46 configuration.
IPv6 map hash table size:	IPv6 map hash table size.
Max mapping entries:	The max mapping entries allowed on the system.

TABLE 7 Display fields for show ip nat statistics (Continued)

This field...	Displays...
Number of free map entries:	The number of free map entries.
NAT64 IPv4 NAT Pools:	Lists all NAT64 pool configuration.
Number of NAT pools configured:	The number of NAT pools configured on the system.
Number of NAT pool IPs configured:	The number of NAT pool IP addresses configured on the system. The maximum number is 192.
Head pool:	Identifier of the first pool configured.
Current pool:	Identifier of the current IPv4 pool being used.
nat64 pool test:	Lists details about the NAT64 IPv4 pools on the system.

Clearing NAT64 information

You can clear the following information from a NAT64 configuration:

- NAT64 session entries from the session table
- NAT64 statistics (stateless and stateful)

Clearing NAT64 session entries from the session table

You can use the **clear nat64 session** command to clear NAT64 session entries from the session table.on the ServerIron ADX. This output is displayed as follows.

```
ServerIron ADX# clear nat64 session
```

Syntax: clear nat64 session

Clearing stateless and stateful NAT64 statistics

You can use the **clear nat64 statistics** command to clear stateless and stateful NAT64 statistics.on a ServerIron ADX. This command is issued as follows.

```
ServerIron ADX# clear nat64 statistics
```

Syntax: clear nat64 statistics

NAT64 Connection logging

A ServerIron ADX provides NAT64 connection logging to enable administrators to audit and log NAT64 connections created on the ServerIron ADX. A user can configure the ServerIron ADX to send a message to an external Syslog server each time NAT64 creates session table entries for NAT64 traffic.

The forward flow for NAT64 is from the IPv6 Client to the NAT64 IPv6 prefix::ipv4 destination address. The ServerIron ADX selects a NAT pool IP and port to replace the Client IP and Client port and strips off the NAT64 prefix to create the IPv4 destination address.

The NAT64 connection logging displays the following information:

- Protocol
- Client IP
- Client Port
- NAT64 prefix
- IPv4 destination IP
- Destination port
- NAT pool IP
- NAT port

A user can recreate an IPv6 destination IP address by pre-pending the NAT64 prefix to the IPv4 destination IP address.

Beginning with release 12.301a, NAT64 connection creation is logged. A Syslog message is sent each time a flow session pair is created. There is no buffering or batching in the current release. The ServerIron ADX does not currently log connection teardown.

NOTE
This feature is only applicable to Stateful NAT64 since no sessions are created for Stateless NAT64 traffic.

NOTE
Enabling NAT64 logging will have an impact on performance.

Configuring NAT64 Connection logging

To enable NAT64 Connection logging on a ServerIron ADX, you must configure the IP address of the external Syslog Server and enable NAT64 connection-logging.

Configuring the IP address of the external Syslog Server

You can configure the IP address of the external Syslog Server using the following commands.

```
ServerIronADX#configure terminal
ServerIronADX(config)# logging 100.100.100.1
ServerIronADX(config)#
```

Syntax: [no] logging <*ip_address*>

Enabling NAT64 connection-logging

You can enable NAT64 connection-logging using the following commands.

```
adx-nat64#conf t
adx-nat64(config)#nat64 connection-log
adx-nat64(config)#
```

Syntax: [no] nat64 connection-log

Example of NAT64 Connection logging

The following example displays Syslog output for NAT64 Connection logging.

```
USER.INFO Jul 13 02:44:47 192.168.13.1 NAT64-EST proto=UDP
sip=2013::20c:29ff:fe06:4473 sp=53947 prefix=3013:: dip=192.168.130.200 dp=00053
nip=192.168.130.10 np=37888
USER.INFO Jul 13 02:44:57 192.168.13.1 NAT64-EST proto=TCP
sip=2013::20c:29ff:fe06:4473 sp=35659 prefix=3013:: dip=192.168.130.200 dp=00053
nip=192.168.130.10 np=37889
```

TABLE 8 Display fields for NAT64 connection logging

This field...	Displays...
USER.INFO	Informational message.
Time stamp	The date and time the message was logged.
NAT64-EST	NAT64 session creation event.
proto	The protocol (UDP or TCP).
sip	The client IP address.
sp	The client port.
prefix	The NAT64 prefix in the destination address.
dip	The IPv4 address in the destination IPv6 address.
dp	The destination port.
nip	The NAT64 NAT pool IP address.
np	The NAT64 NAT port.

Chapter

IPv4-only client to IPv6 resource

3

Overview

A NAT46 gateway enables IPv4-only clients and IPv6 resources to communicate with each other via address and packet translation. This translation is performed in stateless mode on a NAT46 gateway using a mapping table. The mapping table matches IPv4 request packets from the client sent to the gateway to an IPv6 destination address of the IPv6 resource.This mode requires the following components

- NAT64 IPv4 Prefix – The NAT64 IPv4 prefix provides a range of IPv4 addresses on the NAT64 gateway that can be used to represent IPv6 resources. It is created by defining an IPv4 prefix with a subnet mask. Any IPv4 address within the defined subnet can then be assigned to an IPv6 resource that is being made available through the gateway.
- NAT64 IPv6 Prefix – The NAT64 IPv6 prefix is used to create a unique IPv6 address for IPv6 resources it makes available through the gateway. The defined NAT64 IPv6 prefix is concatenated with an IPv4 address selected from within the subnet defined by the NAT64 IPv4-only prefix.
- DNS64 server – This component (not supplied by Brocade) is responsible for providing the IPv4 clients with the IPv4 addresses defined for IPv6 resources that the client wants to utilize. These IPv4 addresses are defined within the range specified by the NAT64 IPv4 prefixes and are assigned by the administrator of the DNS64 server. The DNS64 server also maintains all of the IPv6 addresses of the resources within the IPv6-only network and makes them available to the NAT46 gateway.
- NAT46 gateway – This component is installed on a ServerIron ADX. It is reached by the IPv4 client using an IPv4 address supplied to it by the DNS server. When packets from the IPv4 client reach the NAT46 gateway, they are mapped to an IPv6 address for the resource that they want to access. This mapping is provided through use of a mapping table. This table can be populated by any of the following three methods: static mapping, real-time DNS query, or pre-fetched DNS query.

Populating the NAT46 mapping table

As described, NAT46 uses a mapping table to match the IPv4 destination address defined for an IPv6 service with the actual IPv6 destination address of the service. This table is generated by one of the following methods:

Static Mapping – You can manually configure the NAT46 gateway to contain an entry in the mapping table to translate an IPv4 destination address to an IPv6 destination address. These mappings are defined using the **nat64 map** command.

DNS Dynamic Learning – If a packet is received at the NAT46 gateway with IPv4 destination address within the range defined by the NAT64 IPv4 prefix and it does not contain an entry in its mapping table for that IPv4 address, the gateway will send a PTR query to the DNS64 server to obtain the hostname of the resource it is trying to reach. The NAT46 gateway then sends a query for the hostname to determine the corresponding IPv6 address. The mapping defined by this operation is then entered into the mapping table of the NAT46 gateway.

Pre-fetched DNS Dynamic Learning – Optionally, the NAT46 gateway can be configured (**prefetch** option) to periodically send PRT queries to the DNS64 server (as described for Real-time DNS Query) to determine IPv6 address translations for each of the IPv4 destination addresses defined IPv4 addresses within the range defined by its IPv4 prefix. The NAT46 gateway uses this information to populate its mapping table.

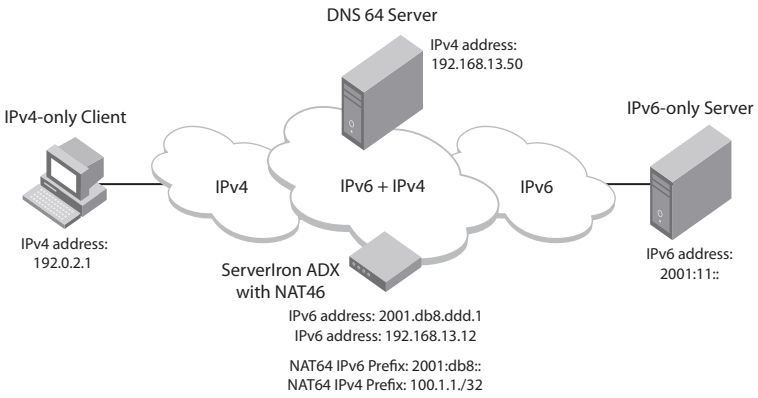
Operation of NAT46 for IPv4-only client to IPv6 resource

Figure 9 provides a high-level view of the IPv4-only client to IPv6 resource configuration. As shown, the client only resides in an IPv4 network and the server (resource) only resides in an IPv6 network. The DNS64 server and the ServerIron ADX (configured as a NAT64 gateway) communicate with both the IPv4 and IPv6 networks.

The ServerIron ADX configured as a NAT46 gateway has an IPv4-prefix that defines the IPv4 addresses that represent the IPv6 resources that it makes available to IPv4 clients. In this example, an IPv4 prefix of 100.1.1./32 is configured on the NAT46 gateway. This means that any IP address within the subnet“100.1.1.x” can be assigned to an IPv6 resource in the IPv6-only network. In this example, the IPv4 address “100.1.1.1” is assigned by the administrator of the DNS64 server to the IPv6 resource “brocadetest.com” (IPv6 address 2001:db8::).

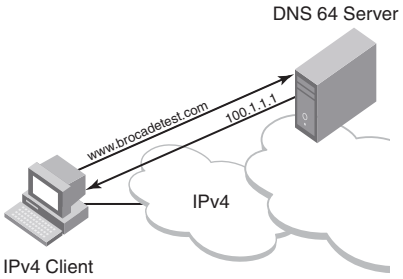
The gateway is also configured with an IPv6-prefix that when combined with a client IPv4 address provides a source IPv6 address on the NAT46 gateway that represents each of the IPv4 clients to the IPv6 resources. The NAT64 IPv6 prefix defined on this gateway is 2001:11::.

FIGURE 9 IPv6-only client to IPv4 resource overview



The DNS64 server (not supplied by Brocade) is configured to respond to a query from the IPv4 client with an IPv4 address within the subnet defined by the IPv4 prefix. In the example shown in Figure 10, the IPv4 client sends a query to the DNS64 server for the IPv4 address of the resource “www.brocadetest.com” and the DNS server responds with the IPv4 address “100.1.1.1”.

FIGURE 10 IPv6 client to DNS server communication

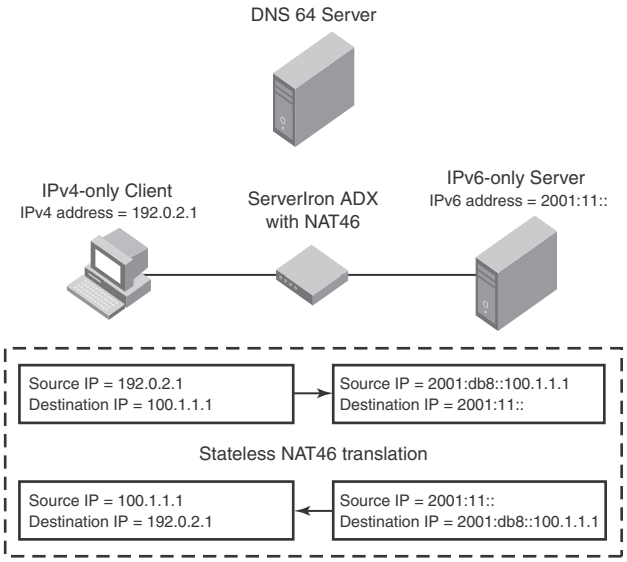


The request packet is then sent with the IPv4 source address of the client (192.0.2.1) to the destination IPv4 address that was obtained for “brocadetest.com” (100.1.1.1) from the DNS64 server. The packet destined for “brocadetest.com” arrives at the NAT46 gateway with a destination address of “100.1.1.1”. The NAT46 gateway then uses its mapping table to translate the destination address into the IPv6 address of the IPv6-only server that hosts “brocadetest.com”. The mapping table used for this translation is populated as described in “Populating the NAT46 mapping table” on page 29. In this example, the IPv6-only resource has an IPv6 address of 2001:11::.

The NAT46 gateway then uses the IPv6-prefix with the IPv4 address that the client used to reach the IPv6 resource to create a source address for the packet being forwarded to the IPv6-only resource. In this example, the IPv6-prefix (2001:db8) is combined with the IPv4 address of the IPv6 resource (100.1.1.1) to create the IPv6 source address “2001:db8 100.1.1.1” for the packet being forwarded to the IPv6 resource.

When the IPv6-only resource sends a return packet to the IPv4-only client, the NAT46 gateway strips out the IPv6 portion of the destination address and uses the IPv4 portion for the source address. The gateway then maps the correct IPv4 destination address as determined from its mapping table. The process occurs as shown in Figure 11.

FIGURE 11 Stateless NAT46 translation



In this example, the IPv4 address (100.1.1.1) is stripped out of the concatenated IPv6 packet and used as the source address in the packet forwarded to the IPv4 client. That IPv4 address is mapped to the clients IPv4 address (192.0.2.1) in the mapping table and used as the destination IPv4 address.

Configuring NAT64 for IPv4-only client to IPv6 resource

Configuration of NAT64 for an IPv6-only client to an IPv4-only resource is performed using the following:

- Configure a NAT64 IPv4 prefix (required)
- Configure a NAT64 IPv6 prefix (required)
- Configuring static mapping (optional)
- Configure DNS dynamic learning (optional)
- Configure a back-off interval for DNS discoveries (optional)

NOTE
The maximum number if NAT prefixes that can be configured on a ServerIron ADX is 8.

NOTE
If you are configuring NAT64 on a system that has had a previous SLB configuration see "When installing NAT64 on a ServerIron ADX with a previous config" on page 6

Configuring a NAT64 IPv4 prefix

A NAT64 IPv4 prefix is configured on a ServerIron ADX as shown.

```
ServerIron ADX(config) nat64 ipv4-prefix 200.1.1.0/24 prefetch
```

Syntax: [**no**] **nat64 ipv4-prefix** <prefix/subnet> [**inject-static-route**] [**prefetch**]

The <prefix/subnet> variable specifies the NAT64 IPv4 prefix that will be used by the ServerIron ADX when operating as a NAT64 gateway.

The **inject-static-route** option injects the host route into the routing protocol. The host route will only be injected if the static map command is issued or a dynamic mapping is found. Unlike when an IPv6 prefix route is injected, the IPv4 route injection configuration does not require you to specify an interface. Route injection for IPv4 uses the null0 route. NAT46 route injection is described in detail in "Route injection NAT46" on page 35.

The **prefetch** option directs the ServerIron ADX to prefetch IPv4 to IPv6 mappings from DNS. the **nat64 dns-dynamic-learning** command must be configured for this option to take effect. If it is not configured, an error message will be displayed.

Configuring a NAT64 IPv6 prefix

A NAT64 IPv6 prefix is configured on a ServerIron ADX as shown.

```
ServerIron ADX(config) nat64 ipv6-prefix 2001:db7:8000::0/96 inject-static-route
ve 7 stateless
```

Syntax: [**no**] **nat64 ipv6-prefix** <prefix/subnet> [**inject-static-route** { **ve** <port-number> | **ethernet** <port-number>] **stateless**

The <prefix/subnet> variable specifies the NAT64 IPv6 prefix that will be used by the ServerIron ADX when operating as a NAT46 gateway.

The **inject-static-route** option is used to advertise the subnet defined by the <prefix/subnet> variable on the IPv6 network. It is configured for an IPv6 interface specified by the **ve** <port-number> or **ethernet** <port-number> variable. It should be the interface connected to the adjacent router. NAT46 route injection is described in detail in "Route injection NAT46" on page 35.

Configuring static mapping

You can populate the mapping table of the NAT46 gateway statically by configuring a **nat64 map** command that identifies an IPv4 address within the subnet defined by the NAT64 IPv4 prefix and maps it to the IPv6 address of a resource. This mapping is performed as shown.

```
ServerIron ADX(config) nat64 map 201.1.1.100 2001:db7:8000::100
```

Syntax: [**no**] **nat64 map** <IPv4-address> <IPv6-address>

The <IPv4-address> variable defines an IPv4 address within the subnet defined by the NAT64 IPv4 prefix that identifies an IPv6 resource to the IPv4 only network.

The <IPv6-address> variable specifies the IPv6 address of the IPv6 resource that is being mapped to the IPv4 address specified by the <IPv4-address> variable within this command.

Configuring DNS dynamic learning

You can configure a ServerIron ADX to perform dynamic learning of IPv4 to IPv6 mappings through DNS as shown.

```
ServerIron ADX(config) nat64 dns-dynamic-learning
```

Syntax: [**no**] **nat64 dns-dynamic-learning**

With this command configured a ServerIron ADX will discover IPv6 to IPv4 mappings through DNS whenever the DNS server receives a new IPv4 destination or IPv6 source. With this command configured, you can configure a ServerIron ADX to prefetch mappings for IPv4 prefixes using the **nat64 ipv4-prefix <prefix/subnet> prefetch** command.

You can clear all entries created using dynamic learning as described in "Clearing IPv6-IPv4 mappings learned through DNS" on page 40.

Configuring a back-off interval for DNS discoveries

A DNS discovery (or refresh) fails if three retries time-out or the DNS64 server returns an error. In this situation the NAT46 gateway may still receive traffic intended for IPv6 resources. Instead of retrying a request to the DNS64 server immediately, you can configure a ServerIron ADX to wait for a period of time. This is configured as shown.

```
ServerIron ADX(config) nat64 dns-fail-holdoff 300
```

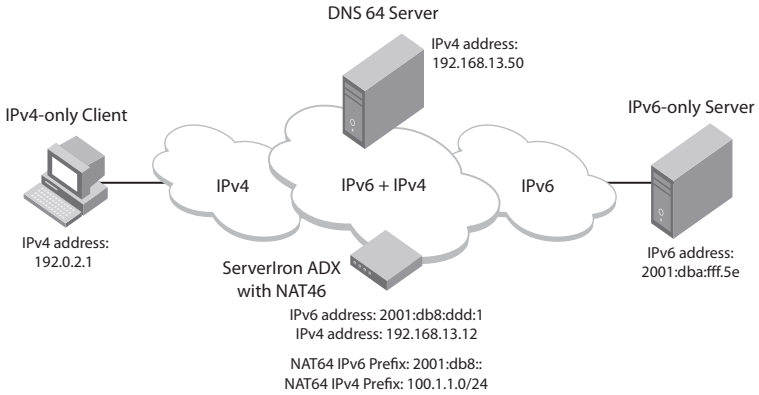
Syntax: [**no**] **nat64 dns-fail-holdoff** <holdoff-interval>

The <holdoff-interval> variable is configured in seconds. The default value is 180 seconds. Configurable values are 10 to 3600 seconds.

IPv4-only client to IPv6 resource sample configuration

The example in Figure 12 is a NAT46 configuration that uses the DNS Dynamic Learning method to populate its translation map.

FIGURE 12 NAT64 stateless example



The following commands are required on a ServerIron ADX to enable the NAT46 stateless configuration shown in Figure 12.

```
ServerIron ADX(config) nat64 dns-dynamic-learning
ServerIron ADX(config) nat64 ipv4-prefix 100.1.1.0/24
ServerIron ADX(config) nat64 ipv6-prefix 2001:db8:: stateless
ServerIron ADX(config) ip dns 192.168.13.50
ServerIron ADX(config-if-1000-6) ip address 192.168.13.12
ServerIron ADX(config-if-1000-6) ipv6 address 2001:db8:ddd.1
```

Configuration Options:

The following options can be added to your configuration as described.

Avoiding IPv6-side fragmentation

To avoid generation of fragments due to increased IP header size with IPv6 when IPv4 sends a full-sized packet (1500 bytes), increase the global IP MTU or the MTU on the IPv6 side interface. If this is not configured, a full-sized IPv4 packet will generate two IPv6 fragments.

Global MTU Configuration

```
ServerIron ADX(config) ip global-mtu 1520
```

IPv6-side MTU Configuration

```
ServerIron ADX(config) interface ve 10
```

Changing the DNS back-off interval

To change the back-off interval for DNS discovery (during which the ServerIron ADX does not try to re-query the DNS server for mapping that recently generated a failure) use the following commands.

```
ServerIron ADX(config) nat64 dns-fail-holdoff 300
```

Pre-fetching IP mappings from the DNS server

To pre-fetch all of the available mappings for the IPv4 subnet defined by the IPv4 prefix, use the prefetch option as shown.

```
ServerIron ADX(config) nat64 ipv4-prefix 100.1.1.0/24 prefetch
```

Configuring a static mapping

You can configure a static mapping between an IPv4 address and an IPv6 address as shown.

```
ServerIron ADX(config) nat64 map 100.1.1.1 /32 2001:db8:fff:32
```

To delete a static mapping, just use the command with the **no** option as shown.

```
ServerIron ADX(config) no nat64 map 100.1.1.1 /32 2001:db8:fff:32
```

Route injection NAT46

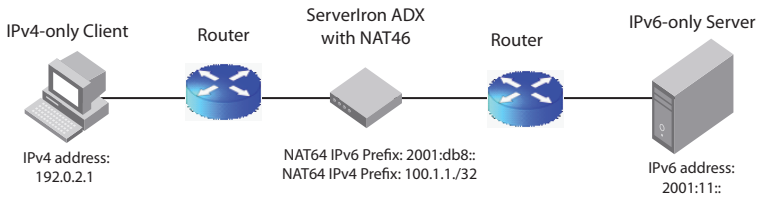
As described previously, IPv4 addresses are defined on the ServerIron ADX to represent IPv6 resources to IPv4 clients. This is done by defining an IPv4 prefix for an IPv4 subnet where the IPv4 addresses can reside. Also, an IPv6 prefix for an IPv6 subnet is defined on the ServerIron ADX to represent IPv4 clients to the IPv6 resources. The addresses assigned for these translations need to be made available as destinations in the routing tables of the IPv6 and IPv4 networks that they reside in. For this reason, we provide command options that direct the ServerIron ADX to inject routes to these addresses into respective networks routing tables. Route distribution is then performed using any of the routing protocols supported by the ServerIron ADX: OSFP, IS-IS and BGP (IPv4 and IPv6).

NAT46 route injection requires that the ServerIron ADX have a routing adjacency relationship with a router for the protocols you want to support route distribution with. For resources that are mapped to an IPv4 address, the IPv4 subnet is defined by the IPv4 prefix specified with the **nat64 ipv4-prefix** command. This command has the **inject-static-route** option which directs the ServerIron ADX to inject a static route to each of the IPv4 addresses within the defined IPv4 subnet that have an IPv6 address mapped to them. This mapping can be either performed using DNS dynamic mapping or though creation of a static entry using the **nat64 map** command. This destination is advertised to the adjacent neighbor and made available in routing tables of the IPv4 network.

For resources that are mapped to a subnet defined by the IPv6 prefix, the **nat64 ipv6-prefix** command has an option to identify an interface on the ServerIron ADX (Ethernet or VE) as the destination for the IPv6 subnet. The interface configured with this command must have an IPv6 address assigned to it.

Figure 13 shows a typical IPv4-only client to IPv6 resource topology configured with router adjacency relationships on both the IPv4 and IPv6 sides of the ServerIron ADX. In this configuration, routes defined by the IP4 prefix and IPv6 prefix are advertised to the adjacent routers and distributed to the respective networks using the routing protocol configured. To accomplish this redistribution you must configure the routing protocol on your ServerIron ADX with a **redistribute static** command.

FIGURE 13 NAT64 Route injection



For more details about how to configure routing protocols on a ServerIron ADX, see the following chapters in the *ServerIron ADX Switch and Router Guide*:

- Configuring OSPF
- Configuring IPv6 Dynamic Routing
- Configuring IS-IS (IPv4)
- Configuring IPv6 IS-IS
- Configuring BGP4 (IPv4
- Configuring BGP4+

NAT46 Route injection example

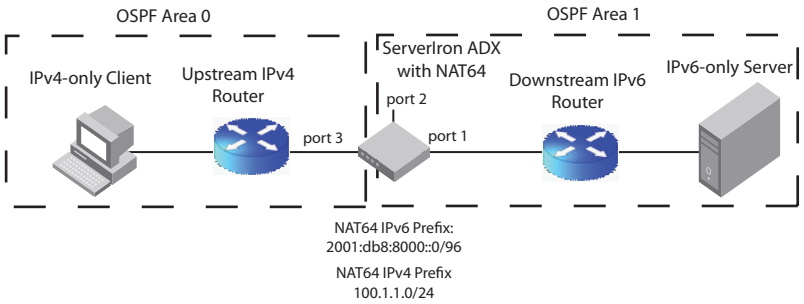
The following example configures a ServerIron ADX with NAT46 for route injection into OSPF with the following required details:

- an IPv6 address is configured: VLAN 1 is configured with VE 7 which has an IPv6 address (2001:db8:8000::1)
- OSPF and OSPFv6 are configured for static route redistribution
- The IPv4 side is configured as OSPF Area 0 and the IPv6 side is configured as OSPF Area 1
- The **nat64 ipv4-prefix** command is configured with the **inject-static-route** option.
- The **nat64 ipv6-prefix** command is configured with the **inject-static-route** option which specifies the **ve 7** interface.
- The **nat64 map** command is configured to map IPv4 address 100.1.1.100 to IPv6 address 2001:db8:8000::100

In a NAT46 configuration, a route to the entire subnet defined by the **nat64 ipv4-prefix** is not injected into the routing table. Only the IPv4 addresses for routes that are mapped are injected. In this example, that means that only a route to 100.1.1.100 will be distributed into the IPv4 network. In cases where the **nat64 dns-dynamic-learning** command is used, routes for all IPv4 addresses that are learned are distributed.

For a description of NAT64 route injection see "Route injection NAT46" on page 35.

FIGURE 14 NAT64 route injection example




```
ServerIron ADX(config)# vlan 1
ServerIron ADX(config-vlan-1)# interface ethernet 1 to 2
ServerIron ADX(config-vlan-1)# interface ve 7
ServerIron ADX(config-vlan-1)# exit
ServerIron ADX(config)# interface ve 7
ServerIron ADX(config-vif-7)# ipv6 address 2001:db8:8000::1
ServerIron ADX(config-vif-7)# exit
ServerIron ADX(config)# router ospf
ServerIron ADX(config-ospf-router)# redistribute static
ServerIron ADX(config-ospf-router)# area 0
ServerIron ADX(config-ospf-router)# exit
ServerIron ADX(config)# ipv6 router ospf
ServerIron ADX(config-ospf6-router)# redistribute static
ServerIron ADX(config-ospf6-router)# area 1
ServerIron ADX(config-ospf6-router)# exit
ServerIron ADX(config)# nat64 ipv4 prefix 100.1.1.0/24 inject-static-route
ServerIron ADX(config)# nat64 ipv6-prefix 2001:db8:8000::0/96 inject-static-route
ve 7 stateless
ServerIron ADX(config)# nat64 map 100.1.1.100 2001:db8:8000::100
```

NOTE
While this example describes NAT46 route injection for the OSPF protocol, you can also use this feature with BGP and IS-IS. Only the routing protocol configurations will differ.

Configuring Packet fragmentation with IPv4-only client to IPv6 resource

Packets from the IPv4 client to the IPv6 server can be too large and need to be split into two IPv6 packets. The following describes the criteria for judging that packets are too large:

Regular packets – IP total length greater than 1480 bytes

Fragmented packets – IP total length greater than 1480 + 8 bytes

If the packets exceed these limitations, one of the following actions will be taken:

1. If the **frag-664-reverse-full-sized-pkt** command is configured, the packet will be split and no further actions will be performed.
2. If the condition in step 1 isn't met, and the DF bit is set at the server, the "frag needed" ICMP error message will be sent.
3. If the conditions in steps 1 and 2 aren't met, the packet will be split.

The **frag-664-reverse-full-sized-pkt** command is configured as shown in the following.

```
ServerIronADX(config)# frag-664-reverse-full-sized-pkt
```

Syntax: [no] **frag-664-reverse-full-sized-pkt**

For more information about NAT64 fragmentation support see "NAT64 fragmentation support" on page 5.

High availability for NAT46

The only high availability mode currently supported with NAT46 feature is the Active-Active HA. Support for this mode is essentially the same as for the IPv6 client to IPv6 resource configuration as described in "High availability for NAT64" on page 17 with the following differences.

- Each ServerIron ADX is configured with a NAT64 IPv6 prefix, specifying the IPv6 address range and a NAT64 IPv4 prefix, specifying the IPv4 address range.
- Because the NAT46 configuration does not use a port pool, the **port pool range** option is not configured.
- Because the NAT46 configuration is stateless, the "inject active only for HA" configuration is not used.

Displaying NAT46 information

You can display the following information for a NAT46 configuration:

- NAT64 IPv4 to IPv6 address mapping
- In-progress NAT64 DNS dynamic learning
- NAT64 statistics – see "Displaying NAT 64 statistics" on page 20
- NAT64 resources – see "Displaying NAT 64 resources" on page 23

Displaying NAT64 IPv4 to IPv6 address mapping

You can use the **show nat64 map** command to display NAT64 IPv4 to IPv6 address mappings in a stateless NAT64 gateway. This output is displayed as follows.

```
ServerIron ADX# show nat64 map all

*****************************
NAT64 IPv4-IPv6 Address Mapping
IPv4 Address    IPv6 Address        Type
1.1.1.1         2001::1             CLI
1.1.1.2         2001::2             DNS
```

```
1.1.1.3                          DNS pending
*****************************
```

Syntax: show nat64 map { <IPv4_address> | <IPv6_address> | **all** }

The <IPv4_address> variable specifies the IPv4 address for the IPv4-IPv6 mapping that you want to display.

The <IPv6_address> variable specifies the IPv6 address for the IPv4-IPv6 mapping that you want to display.

The **all** parameter displays all of the configured Stateless Static NAT64 IPv4-IPv6 address mapping.

TABLE 9 Display fields for show nat map all

This field...	Displays...
IPv4 Address	IPv4 address (Destination for incoming IPv4 packets).
IPv6 Address:	IPv6 address (Source of incoming IPv6 packets).
Type	CLI -> configured DNS -> dynamically learnt PENDING -> dynamic learning on-going

Displaying in-progress NAT64 DNS dynamic learning

You can use the **show nat64 dns-in-flight** command to display NAT64 DNS dynamic learning that is in-progress on a stateless NAT64 gateway. This output is displayed as follows.

```
ServerIron ADX# show nat64 dns-in-flight
NAT64 In-Flight Dynamic Mapping Resolutions:
Type  Tries  DNS Server       Query  IP Address
```

Syntax: show nat64 dns-in-flight

TABLE 10 Display fields for show dns-in-flight

This field...	Displays...
Type	IPv4 -> discovering IPv4 for known IPv6 address. IPv6 -> discovering IPv6 for known IPv4 address.
Tries	# times this query was attempted.
DNS Server	DNS server being used for this query.
Query	DNS query type sent, can be A, AAAA or PTR.
IP Address	Known IP address IP(v4 or IPv6).

Clearing NAT46 information

You can clear the following information from a NAT46 configuration:

- IPv6-IPv4 mappings learned through DNS
- NAT64 statistics (stateless and stateful) – see "Clearing stateless and stateful NAT64 statistics" on page 25

Clearing IPv6-IPv4 mappings learned through DNS

You can use the **clear nat64 dns-dynamic-mapping** command to clear sIPv6-IPv4 mappings learned through DNS on a ServerIron ADX. This command is issued as follows.

```
ServerIron ADX# clear nat64 dns-dynamic-mapping 1.1.1.1
```

Syntax: clear nat64 dns-dynamic-mapping { <IPv4_address> | <IPv6_address> | **all** }

The <IPv4_address> variable specifies the IPv4 address for the IPv4-IPv6 mapping that you want to clear.

The <IPv6_address> variable specifies the IPv6 address for the IPv4-IPv6 mapping that you want to clear.

The **all** parameter clears all of the configured Stateless Static NAT64 IPv4-IPv6 address mapping.

Chapter

Access Control List

4

How ServerIron processes ACLs

This chapter describes the Access Control List (ACL) feature. ACLs allow you to filter traffic based on the information in the IP packet header. Depending on the Brocade device, the device may also support Layer 2 ACLs, which filter traffic based on Lay 2 MAC header fields.

You can use IP ACLs to provide input to other features such as distribution lists and rate limiting. When you use an ACL this way, use permit statements in the ACL to specify the traffic that you want to send to the other feature. If you use deny statements, the traffic specified by the deny statements is not supplied to the other feature.

There are two ways that IPv4 ACLs are processed in Brocade devices: in software and in hardware. This processing differs depending on the software release that you are running. These differences are described in the following sections.

Prior to release 12.3.01

Prior to release 12.3.01, IPv4 ACLs were processed as described in the following:

For deny actions:

All deny packets are dropped in hardware.

For permit actions:

For pass-through traffic, packets are processed in hardware.

For Layer 4 - 7 traffic, packets are forwarded to the BPs and the BPs perform the ACL processing.

Beginning with release 12.3.01 and later

Beginning with release 12.3.01, IPv4 ACLs are processed as described in the following:

For deny actions:

All deny packets are dropped in hardware.

For permit actions:

For pass-through traffic, packets are processed in hardware.

For Layer 4 - 7 traffic, packets are processed in hardware and then forwarded to the BPs. The BPs do not take any action on the ACLs.

Backwards compatibility option:

You can use the **ip flow-based-acl-enable** command to provide backwards compatibility for IPv4 ACL processing. If this command is configured, Layer 4 - 7 traffic, packets are processed in hardware and then forwarded to the BPs where the BPs also process the ACLs. This command is configured as shown in the following.

```
ServerIronADX(config)# ip flow-based-acl-enable
```

Syntax: ip flow-based-acl-enable

Rule-based ACLs

Some Brocade devices process the traffic that ACLs filter in hardware. This document refers to this type of ACLs as rule-based ACLs. These ACLs are programmed into hardware at startup or as a new ACL is entered.

Rule-based ACLs program the ACL entries you assign to an interface into Content Addressable Memory (CAM) space allocated for the port(s). Devices that use rule-based ACLs program the ACLs into the CAM entries and use these entries to permit or deny packets in the hardware, without sending the packets to the CPU for processing.

Rule-based ACLs are supported on physical interfaces ve interfaces and trunk groups.

Configuration guidelines for rule-based ACLs: general guidelines

Consider the following guidelines:

- Rule-based ACLs support only one ACL per port. The ACL of course can contain multiple entries (rules). For example, rule-based ACLs do not support ACLs 101 and 102 on port 1, but rule-based ACLs do support ACL 101 containing multiple entries.
- If you change the content of an ACL (add, change, or delete entries), you must remove and then reapply the ACL to all the ports that use it. Otherwise, the older version of the ACL remains in the CAM and continues to be used. You can easily re-apply ACLs using the **ip rebind-acl** <*num*> | <*name*> | **all** command. Refer to "Applying an ACLs to interfaces" on page 64.

NOTE
Brocade recommends that you also remove and reapply a changed ACL.

- ACL statistics are not supported with rule-based rate limiting.
- If you use the <*icmp-type*> parameter with an extended ACL, the device uses the CPU to filter packets using the ACL. The CPU is required to filter the ICMP message type.
- You can use PBR and rule-based ACLs on the same port. However, Brocade recommends that you use exactly the same ACL for each feature. Otherwise, it is possible for the ACL's Layer 4 CAM entry to be programmed incorrectly and give unexpected results.

How fragmented packets are processed

The descriptions for rule-based ACLs above apply to non-fragmented packets. The default processing of fragments by rule-based ACLs is as follows:

- The first fragment of a packet is permitted or denied using the ACLs. The first fragment is handled the same way as non-fragmented packets, since the first fragment contains the Layer 4 source and destination application port numbers. The device uses the Layer 4 CAM entry if one is programmed, or applies the interface's ACL entries to the packet and permits or denies the packet according to the first matching ACL.
- For other fragments of the same packet, one of the following occurs:
  - If the device has a CAM entry for the packet and has not been configured to send the fragments to the CPU, the device uses the CAM entry to forward the fragments in hardware.

    The fragments are forwarded even if the first fragment, which contains the Layer 4 information, was denied. Generally, denying the first fragment of a packet is sufficient, since a transaction cannot be completed without the entire packet. However, for stricter fragment control, you can send fragments to the CPU for filtering.
  - If the device is configured to send fragments to the CPU for filtering, the device compares the source and destination IP addresses to the ACL entries that contain Layer 4 information.
    - If the fragment's source and destination addresses exactly match an ACL entry that has Layer 4 information, the device assumes that the ACL entry is applicable to the fragment and permits or denies the fragment according to the ACL entry. The device does not compare the fragment to ACL entries that do not contain Layer 4 information.
    - If both the fragment's source and destination addresses do not exactly match an ACL entry, the device skips the ACL entry and compares the packet to the next ACL entry. This is true even if either the source or destination address (but not both) does exactly match an ACL entry.
    - If the source and destination addresses do not exactly match any ACL entry on the applicable interface, the device drops the fragment.

NOTE
By default, 10 Gigabit Ethernet modules also forward the first fragment instead of using the ACLs to permit or deny the fragment.

You can modify the handling of denied fragments. In addition, you can throttle the fragment rate on an interface that used rule-based ACLs. Refer to "Dropping all fragments that exactly match a flow-based ACL" on page 67 and "Enabling ACL filtering of fragmented packets" on page 68.

Default ACL action

The default action when no ACLs is configured on a device is to permit all traffic. However, once you configure an ACL and apply it to a port, the default action for that port is to deny all traffic that is not explicitly permitted on the port:

- If you want to tightly control access, configure ACLs consisting of permit entries for the access you want to permit. The ACLs implicitly deny all other access.
- If you want to secure access in environments with many users, you might want to configure ACLs that consist of explicit deny entries, then add an entry to permit all access to the end of each ACL. The software permits packets that are not denied by the deny entries.

Types of IP ACLs

Rule-based ACLs can be configured as standard or extended ACLs. A standard ACL permits or denies packets based on source IP address. An extended ACL permits or denies packets based on source and destination IP address and also based on IP protocol information.

Standard or extended ACLs can be numbered or named. Standard numbered ACLs have an idea of 1 – 99. Extended numbered ACLs are numbered 100 – 199. IDs for standard or extended ACLs can be a character string. In this document, ACLs with a string ID is called a named ACL.

ACL IDs and entries

ACLs consist of ACL IDs and ACL entries:

- **ACL ID** – An ACL ID is a number from 1 – 99 (for a standard ACL) or 100 – 199 (for an extended ACL) or a character string. The ACL ID identifies a collection of individual ACL entries. When you apply ACL entries to an interface, you do so by applying the ACL ID that contains the ACL entries to the interface, instead of applying the individual entries to the interface. This makes applying large groups of access filters (ACL entries) to interfaces simple.

  NOTE
  This is different from IP access policies. If you use IP access policies, you apply the individual policies to interfaces.

- **ACL entry** – An ACL entry are the filter commands associated with an ACL ID. These are also called “statements”. The maximum number of ACL entries you can configure is a system-wide parameter and depends on the device you are configuring. You can configure up to the maximum number of entries in any combination in different ACLs. The total number of entries in all ACLs cannot exceed the system maximum.
- Layer 3 switch code on devices can support up to 8192 ACL entries.

You configure ACLs on a global basis, then apply them to the incoming or outgoing traffic on specific ports. You can apply only one ACL to a port’s inbound traffic and only one ACL to a port’s outbound traffic. The software applies the entries within an ACL in the order they appear in the ACL’s configuration. As soon as a match is found, the software takes the action specified in the ACL entry (permit or deny the packet) and stops further comparison for that packet.

Support for up to 4096 ACL entries

You can configure up to 4096 ACL entries on devices that have enough space to hold a startup-config file that contains the ACLs.

For system-max configuration of 4096 ACL rules, the Ip access-group max-l4-cam parameter must be set to 4096. To configure the maximum ACL rule limit of 4096 ACL rules, the following must be set:

1. The system-max for Ip-filter-sys value must be set to 4096.

   ```
   ServerIronADX(config)# system-max ip-filter-sys 4096
   ```

2. The Ip access-group max-l4-cam parameter must be set to 4096 on the interface that the ACL will be applied

   ```
   ServerIronADX(config)# interface ethernet 1
   ServerIronADX(config-if-e1000-1)# ip access-group max-l4-cam 4096
   ```

3. Execute the **write memory** command to save the running configuration to the startup-config reload the ServerIron ADX.

The actual number of ACLs you can configure and store in the startup-config file depends on the amount of memory available on the device for storing the startup-config. To store 4096 ACLs in the startup-config file requires at least 250K bytes, which is larger than the space available on a device's flash memory module.

You can load ACLs dynamically by saving them in an external configuration file on flash card or TFTP server, then loading them using one of the following commands.

```
copy tftp running-config <ip-addr> <filename>
ncopy tftp <ip-addr> <from-name> running-config
```

In this case, the ACLs are added to the existing configuration.

ACL entries and the Layer 4 CAM

Rule-based ACLs both use Layer 4 CAM entries.

Aging out of entries in the Layer 4 CAM

On a ServerIron ADX device, the device permanently programs rule-based ACLs into the CAM. The entries never age out.

Displaying the number of Layer 4 CAM entries

To display the number of Layer 4 CAM entries used by each ACL, enter the following command.

```
ServerIronADX(config)# show access-list all

Extended IP access list 100 (Total flows: N/A, Total packets: N/A, Total rule cam
use: 3)
permit udp host 192.168.2.169 any (Flows: N/A, Packets: N/A, Rule cam use: 1)
permit icmp any any (Flows: N/A, Packets: N/A, Rule cam use: 1)
deny ip any any (Flows: N/A, Packets: N/A, Rule cam use: 1)
```

Syntax: show access-list <*acl-num*> **|** <*acl-name*> **| all**

The Rule cam use field lists the number of CAM entries used by the ACL or entry. The number of CAM entries listed for the ACL itself is the total of the CAM entries used by the ACL's entries.

Specifying the maximum number of CAM entries for rule-based ACLs

For rule-based ACLs, you can adjust the allocation of Layer 4 CAM space for use by ACLs, on an IPC or IGC basis and on 10 Gigabit Ethernet modules. The new allocation applies to all the ports managed by the IPC or IGC or 10 Gigabit Ethernet module.

Most ACLs require one CAM entry for each ACL entry (rule). The exception is an ACL entry that matches on more than one TCP or UDP application port. In this case, the ACL entry requires a separate Layer 4 CAM entry for each application port on which the ACL entry matches.

Make sure you specify a maximum that is equal to or greater than the largest number of entries required by an ACL applied to any of the ports managed by the same IPC or IGC. For example, if port 1 will have an ACL that requires 250 entries, make sure 250 is the lowest number of entries you specify for any port on IPC 1 (the IPC that manages ports 1 – 24).

To specify the maximum number of CAM entries the device can allocate for rule-based ACLs, enter commands such as the following.

```
ServerIronADX(config)# interface ethernet 1/1
ServerIronADX(config-if-1/1)# ip access-group max-l4-cam 50
```

This command allows up to 50 ACL entries on each port managed by the IPC or IGC that manages port 1/1.

Syntax: [no] ip access-group max-l4-cam <*num*>

The <*num*> parameter specifies the number of CAM entries and can be from 10 – 2048. The default depends on the device.

The command is valid at the interface configuration level. However, the device applies the change to all ports managed by the same IPC or IGC. Regardless of the port number, when you save the change to the startup-config file, the CLI applies the command to the first port managed by the IPC or IGC. For example, if you enter the command on port 3, when you save the configuration change, the CLI enters the ip access-group max-l4-cam command under port 1 in the startup-config file.

NOTE
If you enter the command on more than one port managed by the same IPC or IGC, the CLI uses the value entered with the most-recent command for all the ports on the ICP or IGC.

Configuring numbered and named ACLs

When you configure ACLs, you can refer to the ACL by a numeric ID or by an alphanumeric name. The commands to configure numbered ACLs are different from the commands for named ACLs:

- If you refer to the ACL by a numeric ID, you can use 1 – 99 for a standard ACL or 100 – 199 for an extended ACL. This document refers to this ACL as *numbered ACL*.
- If you refer to the ACL by a name, you specify whether the ACL is a standard ACL or an extended ACL, then specify the name. This document refers to this ACL type as *named ACL*.

You can configure up to 100 standard numbered IP ACLs and 100 extended numbered IP ACLs. You also can configure up to 100 standard named ACLs and 100 extended named ACLs by number. Regardless of how many ACLs you have, the device can have a maximum of 1024 ACL entries, associated with the ACLs in any combination. (On ServerIron Chassis devices with Management 2 or Management 3 modules, the maximum is 2048.)

Configuring standard numbered ACLs

This section describes how to configure standard numbered ACLs with numeric IDs:

- For configuration information on named ACLs, refer to "Configuring standard or extended named ACLs" on page 54.
- For configuration information on extended ACLs, refer to "Configuring extended numbered ACLs" on page 48.

Standard ACLs permit or deny packets based on source IP address. You can configure up to 99 standard ACLs. There is no limit to the number of ACL entries an ACL can contain except for the system-wide limitation. For the number of ACL entries supported on a device, refer to "ACL IDs and entries" on page 44.

To configure a standard ACL and apply it to outgoing traffic on port 1/1, enter the following commands.

```
ServerIronADX(config)# access-list 1 deny host 209.157.22.26
ServerIronADX(config)# access-list 1 deny 209.157.29.12
ServerIronADX(config)# access-list 1 deny host IPHost1
ServerIronADX(config)# access-list 1 permit any
ServerIronADX(config)# int eth 1/1
ServerIronADX(config-if-1/1)# ip access-group 1 in
ServerIronADX(config)# write memory
```

The commands in this example configure an ACL to deny packets from three source IP addresses from being forwarded on port 1/1. The last ACL entry in this ACL permits all packets that are not explicitly denied by the first three ACL entries.

Standard ACL syntax

Syntax: [no] access-list <*num*> **deny | permit** <*source-ip*> **|** <*hostname*> <*wildcard*>

or

Syntax: [no] access-list <*num*> **deny | permit** <*source-ip*>**/**<*mask-bits*> **|** <*hostname*>

Syntax: [no] access-list <*num*> **deny | permit host** <*source-ip*> **|** <*hostname*>

Syntax: [no] access-list <*num*> **deny | permit any**

Syntax: [no] ip access-group <*num*> **in | out**

The <*num*> parameter is the access list number and can be from 1 – 99.

The **deny | permit** parameter indicates whether packets that match a policy in the access list are denied (dropped) or permitted (forwarded).

The <*source-ip*> parameter specifies the source IP address. Alternatively, you can specify the host name.

NOTE
To specify the host name instead of the IP address, the host name must be configured using the Brocade device's DNS resolver. To configure the DNS resolver name, use the **ip dns server-address...** command at the global CONFIG level of the CLI.

The <*wildcard*> parameter specifies the mask value to compare against the host address specified by the <*source-ip*> parameter. The <*wildcard*> is a four-part value in dotted-decimal notation (IP address format) consisting of ones and zeros. Zeros in the mask mean the packet's source address must match the <*source-ip*>. Ones mean any value matches. For example, the <*source-ip*> and <*wildcard*> values 209.157.22.26 0.0.0.255 mean that all hosts in the Class C sub-net 209.157.22.x match the policy.

If you prefer to specify the wildcard (mask value) in CIDR format, you can enter a forward slash after the IP address, then enter the number of significant bits in the mask. For example, you can enter the CIDR equivalent of "209.157.22.26 0.0.0.255" as "209.157.22.26/24". The CLI automatically converts the CIDR number into the appropriate ACL mask (where zeros instead of ones are the

significant bits) and changes the non-significant portion of the IP address into ones. For example, if you specify 209.157.22.26/24 or 209.157.22.26 0.0.0.255, then save the changes to the startup-config file, the value appears as 209.157.22.0/24 (if you have enabled display of sub-net lengths) or 209.157.22.0 0.0.0.255 in the startup-config file.

If you enable the software to display IP sub-net masks in CIDR format, the mask is saved in the file in
"/<*mask-bits*>" format. To enable the software to display the CIDR masks, enter the **ip show-subnet-length** command at the global CONFIG level of the CLI. You can use the CIDR format to configure the ACL entry regardless of whether the software is configured to display the masks in CIDR format.

NOTE
If you use the CIDR format, the ACL entries appear in this format in the running-config and startup-config files, but are shown with sub-net mask in the display produced by the **show ip access-list** command.

The **host** <*source-ip*> | <*hostname*> parameter lets you specify a host IP address or name. When you use this parameter, you do not need to specify the mask. A mask of all zeros (0.0.0.0) is implied.

The **any** parameter configures the policy to match on all host addresses.

The in | out parameter specifies whether the ACL applies to incoming traffic or outgoing traffic on the interface to which you apply the ACL. You can apply the ACL to an Ethernet port. Note that the out option is not supported in the rule-based ACL mode.

Configuring extended numbered ACLs

This section describes how to configure extended numbered ACLs:

- For configuration information on named ACLs, refer to "Configuring numbered and named ACLs" on page 46.
- For configuration information on standard ACLs, refer to "Configuring standard numbered ACLs" on page 46.

Extended ACLs let you permit or deny packets based on the following information:

- IP protocol
- Source IP address or host name
- Destination IP address or host name
- Source TCP or UDP port (if the IP protocol is TCP or UDP)
- Destination TCP or UDP port (if the IP protocol is TCP or UDP)

The IP protocol can be one of the following well-known names or any IP protocol number from 0 – 255:

- Internet Control Message Protocol (ICMP)
- Internet Group Management Protocol (IGMP)
- Internet Gateway Routing Protocol (IGRP)
- Internet Protocol (IP)
- Open Shortest Path First (OSPF)
- Transmission Control Protocol (TCP)

- User Datagram Protocol (UDP)

For TCP and UDP, you also can specify a comparison operator and port name or number. For example, you can configure a policy to block web access to a specific website by denying all TCP port 80 (HTTP) packets from a specified source IP address to the website's IP address.

To configure an extended access list that blocks all Telnet traffic received on port 1/1 from IP host 209.157.22.26, enter the following commands.

```
ServerIronADX(config)# access-list 101 deny tcp host 209.157.22.26 any eq telnet
ServerIronADX(config)# access-list 101 permit ip any any
ServerIronADX(config)# int eth 1/1
ServerIronADX(config-if-1/1)# ip access-group 101 in
ServerIronADX(config)# write memory
```

Here is another example of commands for configuring an extended ACL and applying it to an interface. These examples show many of the syntax choices.

```
ServerIronADX(config)# access-list 102 perm icmp 209.157.22.0/24
209.157.21.0/24
ServerIronADX(config)# access-list 102 deny igmp host rkwong 209.157.21.0/24
ServerIronADX(config)# access-list 102 deny igrp 209.157.21.0/24 host rkwong
ServerIronADX(config)# access-list 102 deny ip host 209.157.21.100 host
209.157.22.1
ServerIronADX(config)# access-list 102 deny ospf any any
ServerIronADX(config)# access-list 102 permit ip any any
```

The first entry permits ICMP traffic from hosts in the 209.157.22.x network to hosts in the 209.157.21.x network.

The second entry denies IGMP traffic from the host device named "rkwong" to the 209.157.21.x network.

The third entry denies IGRP traffic from the 209.157.21.x network to the host device named "rkwong".

The fourth entry denies all IP traffic from host 209.157.21.100 to host 209.157.22.1.

The fifth entry permits all packets that are not explicitly denied by the other entries. Without this entry, the ACL would deny all incoming or outgoing IP traffic on the ports to which you assign the ACL.

The following commands apply ACL 102 to the incoming traffic on port 1/2 and to the incoming traffic on port 4/3.

```
ServerIronADX(config)# int eth 1/2
ServerIronADX(config-if-1/2)# ip access-group 102 in
ServerIronADX(config-if-1/2)# exit
ServerIronADX(config)# int eth 4/3
ServerIronADX(config-if-4/3)# ip access-group 102 in
ServerIronADX(config)# write memory
```

Here is another example of an extended ACL.

```
ServerIronADX(config)# access-list 103 deny tcp 209.157.21.0/24 209.157.22.0/24
ServerIronADX(config)# access-list 103 deny tcp 209.157.21.0/24 eq ftp
209.157.22.0/24
ServerIronADX(config)# access-list 103 deny tcp 209.157.21.0/24 209.157.22.0/24 lt
telnet neq 5
ServerIronADX(config)# access-list 103 deny udp any range 5 6 209.157.22.0/24
range 7 8
ServerIronADX(config)# access-list 103 permit ip any any
```

The first entry in this ACL denies TCP traffic from the 209.157.21.x network to the 209.157.22.x network.

The second entry denies all FTP traffic from the 209.157.21.x network to the 209.157.22.x network.

The third entry denies TCP traffic from the 209.157.21.x network to the 209.157.22.x network, if the TCP port number of the traffic is less than the well-known TCP port number for Telnet (23), and if the TCP port is not equal to 5. Thus, TCP packets whose TCP port numbers are 5 or are greater than 23 are allowed.

The fourth entry denies UDP packets from any source to the 209.157.22.x network, if the UDP port number from the source network is 5 or 6 and the destination UDP port is 7 or 8.

The fifth entry permits all packets that are not explicitly denied by the other entries. Without this entry, the ACL would deny all incoming or outgoing IP traffic on the ports to which you assign the ACL.

The following commands apply ACL 103 to the incoming traffic on ports 2/1 and 2/2.

```
ServerIronADX(config)# int eth 2/1
ServerIronADX(config-if-2/1)# ip access-group 103 in
ServerIronADX(config-if-2/1)# exit
ServerIronADX(config)# int eth 2/2
ServerIronADX(config-if-2/2)# ip access-group 103 in
ServerIronADX(config)# write memory
```

Extended ACL syntax

Use the following syntax for configuring extended numbered ACLs.

Syntax: [no] **access-list** <*num*> **deny | permit** <*ip-protocol*> <*source-ip*> **|** <*hostname*> <*wildcard*> [<*operator*> <*source-tcp/udp-port*>] <*destination-ip*> **|** <*hostname*> [<*icmp-type*> **|** <*icmp-num*> **|** <*icmp-type-number*> <*icmp-code-number*>] <*wildcard*> [<*operator*> <*destination-tcp/udp-port*>] **[established] [precedence** <*name*> **|** <*num*>] **[tos** <*name*> **|** <*num*>] **[ip-pkt-len** <*value*>]

Syntax: [no] **access-list** <*num*> **deny | permit host** <*ip-protocol*> **any any**

Syntax: [no] **ip access-group** <*num*> **in | out**

The <*num*> parameter indicates the ACL number and be from 100 – 199 for an extended ACL.

The **deny | permit** parameter indicates whether packets that match the policy are dropped or forwarded.

The <*ip-protocol*> parameter indicates the type of IP packet you are filtering. You can specify a well-known name for any protocol whose number is less than 255. For other protocols, you must enter the number. Enter "?" instead of a protocol to list the well-known names recognized by the CLI.

The <*source-ip*> | <*hostname*> parameter specifies the source IP host for the policy. If you want the policy to match on all source addresses, enter **any**.

The <*wildcard*> parameter specifies the portion of the source IP host address to match against. The <*wildcard*> is a four-part value in dotted-decimal notation (IP address format) consisting of ones and zeros. Zeros in the mask mean the packet's source address must match the <*source-ip*>. Ones mean any value matches. For example, the <*source-ip*> and <*wildcard*> values 209.157.22.26 0.0.0.255 mean that all hosts in the Class C sub-net 209.157.22.x match the policy.

If you prefer to specify the wildcard (mask value) in Classless Interdomain Routing (CIDR) format, you can enter a forward slash after the IP address, then enter the number of significant bits in the mask. For example, you can enter the CIDR equivalent of "209.157.22.26 0.0.0.255" as "209.157.22.26/24". The CLI automatically converts the CIDR number into the appropriate ACL mask (where zeros instead of ones are the significant bits) and changes the non-significant portion of the IP address into zeros. For example, if you specify 209.157.22.26/24 or 209.157.22.26 0.0.0.255, then save the changes to the startup-config file, the value appears as 209.157.22.0/24 (if you have enabled display of sub-net lengths) or 209.157.22.0 0.0.0.255 in the startup-config file.

If you enable the software to display IP sub-net masks in CIDR format, the mask is saved in the file in "/<*mask-bits*>" format. To enable the software to display the CIDR masks, enter the **ip show-subnet-length** command at the global CONFIG level of the CLI. You can use the CIDR format to configure the ACL entry regardless of whether the software is configured to display the masks in CIDR format.

NOTE
If you use the CIDR format, the ACL entries appear in this format in the running-config and startup-config files, but are shown with sub-net mask in the display produced by the **show ip access-list** command.

The <*destination-ip*> | <*hostname*> parameter specifies the destination IP host for the policy. If you want the policy to match on all destination addresses, enter **any**.

The <*icmp-type*> | <*icmp-num*> parameter specifies the ICMP protocol type.

- This parameter applies only if you specified **icmp** as the <*ip-protocol*> value.
- If you use this parameter, the ACL entry is sent to the CPU for processing.
- If you do not specify a message type, the ACL applies to all types of ICMP messages.

The <*icmp-num*> parameter can be a value from 0 – 255.

The <*icmp-type*> parameter can have one of the following values, depending on the software version the device is running:

- any-icmp-type
- echo
- echo-reply
- information-request
- log
- mask-reply
- mask-request
- parameter-problem

- redirect
- source-quench
- time-exceeded
- timestamp-reply
- timestamp-request
- unreachable
- <*num*>

The <*operator*> parameter specifies a comparison operator for the TCP or UDP port number. This parameter applies only when you specify **tcp** or **udp** as the IP protocol. For example, if you are configuring an entry for HTTP, specify **tcp eq http**. You can enter one of the following operators:

- **eq** – The policy applies to the TCP or UDP port name or number you enter after eq.
- **gt** – The policy applies to TCP or UDP port numbers greater than the port number or the numeric equivalent of the port name you enter after gt.
- **lt** – The policy applies to TCP or UDP port numbers that are less than the port number or the numeric equivalent of the port name you enter after lt.
- **neq** – The policy applies to all TCP or UDP port numbers except the port number or port name you enter after neq.
- **range** – The policy applies to all TCP or UDP port numbers that are between the first TCP or UDP port name or number and the second one you enter following the range parameter. The range includes the port names or numbers you enter. For example, to apply the policy to all ports between and including 23 (Telnet) and 53 (DNS), enter the following: **range 23 53**. The first port number in the range must be lower than the last number in the range.
- **established** – This operator applies only to TCP packets. If you use this operator, the policy applies to TCP packets that have the ACK (Acknowledgment) or RST (Reset) bits set on (set to "1") in the Control Bits field of the TCP packet header. Thus, the policy applies only to established TCP sessions, not to new sessions. Refer to Section 3.1, "Header Format", in RFC 793 for information about this field.

  NOTE
  This operator applies only to destination TCP ports, not source TCP ports.

The <*tcp/udp-port*> parameter specifies the TCP or UDP port number or well-known name. You can specify a well-known name for any application port whose number is less than 1024. For other application ports, you must enter the number. Enter "?" instead of a port to list the well-known names recognized by the CLI.

The **in | out** parameter specifies whether the ACL applies to incoming traffic or outgoing traffic on the interface to which you apply the ACL. You can apply the ACL to an Ethernet port.

NOTE
The **out** option is not supported in the rule-based ACL mode.

The **precedence** <*name*> | <*num*> parameter of the **ip access-list** command specifies the IP precedence. The precedence option for of an IP packet is set in a three-bit field following the four-bit header-length field of the packet's header. You can specify one of the following:

- **critical** or **5** – The ACL matches packets that have the critical precedence. If you specify the option number instead of the name, specify number 5.

- **flash** or **3** – The ACL matches packets that have the flash precedence. If you specify the option number instead of the name, specify number 3.
- **flash-override** or **4** – The ACL matches packets that have the flash override precedence. If you specify the option number instead of the name, specify number 4.
- **immediate** or **2** – The ACL matches packets that have the immediate precedence. If you specify the option number instead of the name, specify number 2.
- **internet** or **6** – The ACL matches packets that have the internetwork control precedence. If you specify the option number instead of the name, specify number 6.
- **network** or **7** – The ACL matches packets that have the network control precedence. If you specify the option number instead of the name, specify number 7.
- **priority** or **1** – The ACL matches packets that have the priority precedence. If you specify the option number instead of the name, specify number 1.
- **routine** or **0** – The ACL matches packets that have the routine precedence. If you specify the option number instead of the name, specify number 0.

The **tos** <*name*> | <*num*> parameter of the **ip access-list** command specifies the IP ToS. You can specify one of the following:

- **max-reliability** or **2** – The ACL matches packets that have the maximum reliability ToS. The decimal value for this option is 2.
- **max-throughput** or **4** – The ACL matches packets that have the maximum throughput ToS. The decimal value for this option is 4.
- **min-delay** or **8** – The ACL matches packets that have the minimum delay ToS. The decimal value for this option is 8.
- **min-monetary-cost** or **1** – The ACL matches packets that have the minimum monetary cost ToS. The decimal value for this option is 1.

  NOTE
  This value is not supported on 10 Gigabit Ethernet modules.

- **normal** or **0** – The ACL matches packets that have the normal ToS. The decimal value for this option is 0.
- <*num*> – A number from 0 – 15 that is the sum of the numeric values of the options you want. The ToS field is a four-bit field following the Precedence field in the IP header. You can specify one or more of the following. To select more than one option, enter the decimal value that is equivalent to the sum of the numeric values of all the ToS options you want to select. For example, to select the **max-reliability** and **min-delay** options, enter number 10. To select all options, select 15.

The **ip-pkt-len** <*value*> parameter filters ICMP packets based on the IP packet length. The device uses the <*value*> to match the total length field in the IP header of ICMP packets. You can specify a value from 1 – 65535.

NOTE
This parameter applies only if you specified **icmp** as the <*ip-protocol*> value.

The **log** parameter enables SNMP traps and Syslog messages for packets denied by the ACL.

You can enable logging on ACLs and filters that support logging even when the ACLs and filters are already in use. To do so, re-enter the **ACL** or **filter** command and add the **log** parameter to the end of the ACL or filter. The software replaces the **ACL** or **filter** command with the new one. The new ACL or filter, with logging enabled, takes effect immediately.

Configuring standard or extended named ACLs

To configure a named IP ACL, use the following CLI method.

The commands for configuring named ACL entries are different from the commands for configuring numbered ACL entries. The command to configure a numbered ACL is **access-list**. The command for configuring a named ACL is **ip access-list**. In addition, when you configure a numbered ACL entry, you specify all the command parameters on the same command. When you configure a named ACL, you specify the ACL type (standard or extended) and the ACL number with one command, which places you in the configuration level for that ACL. Once you enter the configuration level for the ACL, the command syntax is the same as the syntax for numbered ACLs.

The following examples show how to configure a named standard ACL entry and a named extended ACL entry.

Configuration example for standard ACL

To configure a named standard ACL entry, enter commands such as the following.

```
ServerIronADX(config)# ip access-list standard Net1
ServerIronADX(config-std-nacl)# deny host 209.157.22.26 log
ServerIronADX(config-std-nacl)# deny 209.157.29.12 log
ServerIronADX(config-std-nacl)# deny host IPHost1 log
ServerIronADX(config-std-nacl)# permit any
ServerIronADX(config-std-nacl)# exit
ServerIronADX(config)# int eth 1/1
ServerIronADX(config-if-1/1)# ip access-group Net1 out
```

The commands in this example configure a standard ACL named “Net1”. The entries in this ACL deny packets from three source IP addresses from being forwarded on port 1/1. Since the implicit action for an ACL is “deny”, the last ACL entry in this ACL permits all packets that are not explicitly denied by the first three ACL entries. For an example of how to configure the same entries in a numbered ACL, refer to “Configuring standard numbered ACLs” on page 46.

Notice that the command prompt changes after you enter the ACL type and name. The “std” in the command prompt indicates that you are configuring entries for a standard ACL. For an extended ACL, this part of the command prompt is “ext“. The “nacl” indicates that are configuring a named ACL.

Syntax: ip access-list extended | standard <*string*> **|** <*num*>

The **extended** | **standard** parameter indicates the ACL type.

The <*string*> parameter is the ACL name. You can specify a string of up to 256 alphanumeric characters. You can use blanks in the ACL name if you enclose the name in quotation marks (for example, “ACL for Net1”). The <*num*> parameter allows you to specify an ACL number if you prefer. If you specify a number, you can specify from 1 – 99 for standard ACLs or 100 – 199 for extended ACLs.

NOTE

For convenience, the software allows you to configure numbered ACLs using the syntax for named ACLs. The software also still supports the older syntax for numbered ACLs. Although the software allows both methods for configuring numbered ACLs, numbered ACLs are always formatted in the startup-config and running-config files in using the older syntax, as follows.

```
access-list 1 deny host 209.157.22.26
access-list 1 deny 209.157.22.0 0.0.0.255
access-list 1 permit any
access-list 101 deny tcp any any eq http
```

The options at the ACL configuration level and the syntax for the **ip access-group** command are the same for numbered and named ACLs and are described in "Configuring standard numbered ACLs" on page 46.

Configuration example for extended ACL

To configure a named extended ACL entry, enter commands such as the following.

```
ServerIronADX(config)# ip access-list extended "block Telnet"
ServerIronADX(config-ext-nacl)# deny tcp host 209.157.22.26 any eq telnet
ServerIronADX(config-ext-nacl)# permit ip any any
ServerIronADX(config-ext-nacl)# exit
ServerIronADX(config)# int eth 1/1
ServerIronADX(config-if-1/1)# ip access-group "block Telnet" in
```

The options at the ACL configuration level and the syntax for the **ip access-group** command are the same for numbered and named ACLs and are described in "Configuring extended numbered ACLs" on page 48.

Displaying ACL definitions

To display the ACLs configured on a device, use the **show ip access-lists** command. Here is an example.

```
ServerIronADX(config)# show ip access-lists
Extended IP access list 101
    deny tcp host 209.157.22.26 host 209.157.22.26 eq http
```

Syntax: show ip access-lists [<*num*>]

The **show access-list** and **show ip access-list** commands have been updated to display ACL entries with line numbers.

Numbered ACL

For a numbered ACL, you can enter a command such as the following.

```
ServerIronADX(config)# show access-list 99 3
Standard IP access-list 99
deny 10.10.10.1
deny 192.168.1.13
permit any
```

Syntax: show access-list <*acl-number*> [<*line-number*>]

Enter the ACL' number for the <*acl-number*> parameter.

Determine from which line you want the displayed information to begin and enter that number for the <*line-number*> parameter.

Named ACL

For a named ACL, enter a command such as the following.

```
ServerIronADX(config)# ip show access-list standard melon 3
Standard IP access-list melon
deny host 5.6.7.8
deny 192.168.12.3
permit any
```

Syntax: show ip access-list <*acl-name*> | <*acl-number*> [<*line-number*>]

Enter the ACL name for the <*acl-name*> parameter or the ACL's number for <*acl-number*>.

Determine from which line you want the displayed information to begin and enter that number for the <*line-number*> parameter.

Displaying ACLs using keywords

You limit the displayed ACL entries to those that contain a specified keyword.

Numbered ACL

You may have the following numbered ACL.

```
ServerIronADX(config)# show access-list 99
Standard IP access-list 99
deny host 1.2.3.4
permit host 5.6.7.8
permit host 5.10.11.12
permit any
```

If you want to display ACL entries beginning with the entry that contains the keyword "5" enter the following command.

```
ServerIronADX(config)# show access-list 99 | begin 5
Standard IP access-list 99
permit host 5.6.7.8
permit host 5.10.11.12
permit any
```

Since the second entry is the first entry that contains the keyword "5", the display begins with line 2.

If you want to display only the ACL entries that contain the keyword "5" enter the following command.

```
ServerIronADX(config)#show access-list 99 | include 5
Standard IP access-list 99
permit host 5.6.7.8
permit host 5.10.11.12
```

The second and third entries in the ACL contain the keyword "5" and are displayed in the **show access-list**.

If you want to exclude ACL entries that contain a keyword from the show access-list display, enter the following command.

```
ServerIronADX(config)# show access-list 99 | exclude 5
Standard IP access-list 99
deny host 1.2.3.4
permit any
```

The second and third ACL entries are left out from the display.

Syntax: show access-list <*acl-number*> **| begin | exclude | include** <*keyword*>

Enter the ACL number for the <*acl-number*> parameter.

Use the **|** operator to indicate a keyword.

Enter the **begin** <*keyword*> parameter to start the display beginning with the first line containing the text that matches the keyword. For example, if you enter `begin Total`, the displayed information begins with the line containing the word "Total".

Enter the **exclude** <*keyword*> parameter to exclude any lines containing text that match the keyword. For example, if you enter `exclude Total`, any line containing the word "Total" is excluded from the display.

Enter the **include** <*keyword*> display only those lines containing text that match the keyword. For example, if you enter `include Total`, any line containing the word "Total" is included in the display.

Named ACLs

You may have the following numbered ACL.

```
ServerIronADX(config)# show access-list melon
Standard IP access-list melon
deny host 1.2.3.4
permit host 5.6.7.8
permit host 5.10.11.12
permit any
```

If you want to display ACL entries beginning with the entry that contains the keyword "5" enter the following command.

```
ServerIronADX(config)# show access-list melon | begin 5
Standard IP access-list melon
permit host 5.6.7.8
permit host 5.10.11.12
permit any
```

Since the second entry is the first entry that contains the keyword "5", the display begins with line 2.

If you want to display only the ACL entries that contain the keyword "5" enter the following command.

```
ServerIronADX(config)# show access-list melon | include 5
Standard IP access-list melon
permit host 5.6.7.8
permit host 5.10.11.12
```

The second and third entries in the ACL contain the keyword "5" and are displayed in the **show access-list**.

If you want to exclude ACL entries that contain a keyword from the show access-list display, enter the following command.

```
ServerIronADX(config)# show access-list melon | exclude 5
Standard IP access-list melon
deny host 1.2.3.4
permit any
```

The second and third ACL entries are left out from the display.

Syntax: show ip access-list <*acl-number*> **| begin | exclude | include** <*keyword*>

Enter the ACL's number for the <*acl-number*> parameter.

Use the **|** operator to indicate a keyword.

Enter the **begin** <*keyword*> parameter to start the display beginning with the first line containing text that matches the keyword. For example, if you enter begin Total, the displayed information begins with the line containing the word "Total".

Enter the **exclude** <*keyword*> parameter to exclude any lines containing text that match the keyword. For example, if you enter exclude Total, any line containing the word "Total" is excluded from the display.

Enter the **include** <*keyword*> display only those lines containing text that match the keyword. For example, if you enter include Total, any line containing the word "Total" is included in the display.

```
If ACL entries, for both numbered and named ACLs, have remarks, the display
will also include the remarks if they contain characters that match the
specified keywords. For example, ACL 99 might contain the following entries:

ServerIronADX(config)# show access-list 99
Standard IP access-list 99
   ACL Remark: Deny Building A
deny host 1.2.3.4
Permit First Floor Users
permit host 5.6.7.8
deny host 5.10.11.12
permit any

To show all entries containing the keyword "deny" you obtain the following
output:

ServerIronADX(config)#show access-list 99 | include deny
Standard IP access-list 99
   ACL Remark: Deny Building A
deny host 1.2.3.4
deny host 5.10.11.12
```

NOTE
All lines with the keyword "deny", including remarks are included in the display.

Modifying ACLs

When you use the Brocade device's CLI to configure any ACL, the software places the ACL entries in the ACL in the order you enter them. For example, if you enter the following entries in the order shown below, the software always applies the entries to traffic in the same order.

```
ServerIronADX(config)# access-list 1 deny 209.157.22.0/24
ServerIronADX(config)# access-list 1 permit 209.157.22.26
```

If a packet matches the first ACL entry in this ACL and is therefore denied, the software does not compare the packet to the remaining ACL entries. In this example, packets from host 209.157.22.26 will always be dropped, even though packets from this host match the second entry.

You can use the CLI to reorder entries within an ACL by individually removing the ACL entries and then re-adding them. To use this method, enter "no" followed by the command for an ACL entry, and repeat this for each ACL entry in the ACL you want to edit. After removing all the ACL entries from the ACL, re-add them.

This method works well for small ACLs such as the example above, but can be impractical for ACLs containing many entries. Therefore, Brocade devices provide an alternative method. The alternative method lets you upload an ACL list from a TFTP server and replace the ACLs in the device's running-config file with the uploaded list. Thus, to change an ACL, you can edit the ACL on the file server, then upload the edited ACL to the device. You then can save the changed ACL to the device's startup-config file.

ACL lists contain only the ACL entries themselves, not the assignments of ACLs to interfaces. You must assign the ACLs on the device itself.

NOTE
The only valid commands that are valid in the ACL list are the **access-list** and **end** commands. The Brocade device ignores other commands in the file.

To modify an ACL by configuring an ACL list on a file server.

1. Use a text editor to create a new text file. When you name the file, use 8.3 format (up to eight characters in the name and up to three characters in the extension).

   NOTE
   Make sure the Brocade device has network access to the TFTP server.

2. Optionally, clear the ACL entries from the ACLs you are changing by placing commands such as the following at the top of the file.

   ```
   no access-list 1
   no access-list 101
   ```

   When you load the ACL list into the device, the software adds the ACL entries in the file after any entries that already exist in the same ACLs. Thus, if you intend to entirely replace an ACL, you must use the **no access-list** <*num*> command to clear the entries from the ACL before the new ones are added.

3. Place the commands to create the ACL entries into the file. The order of the separate ACLs does not matter, but the order of the entries within each ACL is important. The software applies the entries in an ACL in the order they are listed within the ACL. Here is an example of some ACL entries.

   ```
   access-list 1 deny host 209.157.22.26 log
   access-list 1 deny 209.157.22.0 0.0.0.255 log
   access-list 1 permit any
   access-list 101 deny tcp any any eq http log
   ```

   The software will apply the entries in ACL 1 in the order shown and stop at the first match. Thus, if a packet is denied by one of the first three entries, the packet will not be permitted by the fourth entry, even if the packet matches the comparison values in this entry.

4. Enter the command **end** on a separate line at the end of the file. This command indicates to the software that the entire ACL list has been read from the file.

5. Save the text file.

6. On the Brocade device, enter the following command at the Privileged EXEC level of the CLI.

```
copy tftp running-config <tftp-ip-addr> <filename>
```

NOTE
This command will be unsuccessful if you place any commands other than access-list and end (at the end only) in the file. These are the only commands that are valid in a file you load using the **copy tftp running-config...** command.

7. To save the changes to the device's startup-config file, enter the following command at the Privileged EXEC level of the CLI.

```
write memory
```

Here is a complete example of an ACL configuration file.

```
no access-list 1
no access-list 101
access-list 1 deny host 209.157.22.26 log
access-list 1 deny 209.157.22.0 0.0.0.255 log
access-list 1 permit any
access-list 101 deny tcp any any eq http log
end
```

NOTE
Do not place other commands in the file. The Brocade device reads only the ACL information in the file and ignores other commands, including **ip access-group** commands. To assign ACLs to interfaces, use the CLI.

Adding, inserting, replacing, or deleting a comment

You can add, insert, replace, or delete comments to an ACL entry. First enter a show command as discussed in "Displaying a list of ACL entries" on page 63 to determine the line number of the entry you want to update or where you want to insert the new ACL entry. Then enter a command as shown in one of the two sections below.

Numbered ACL: adding or replacing a comment

To add a comment to an ACL entry in a numbered ACL, do the following.

1. Use the **show access-list** to display the entries in an ACL.

Example

```
ServerIronADX(config)# show access-list 99
Standard IP access-list 99
deny host 1.2.4.5
permit host 5.6.7.8
permit any
```

2. To add the comment "Permit all users" to the second entry in the list, enter a command such as the following.

```
ServerIronADX(config)# access-list 99 insert 2 remark Permit all users
```

3. Entering a **show access-list** command displays the following.

```
ServerIronADX(config)# show access-list 99
Standard IP access-list 99
deny host 1.2.4.5
   Permit all users
permit host 5.6.7.8
permit any
```

Syntax: **access-list** <*acl-num*> **insert** <*line-number*> **| replace** <*line-number*> **remark** <*comment-text*>

Simply entering **access-list** <*acl-num*> **remark** <*comment-text*> adds a remark to the next ACL entry you create.

The **insert** <*line-number*> parameter indicates into which entry the comment is to be added.

The **replace** <*line-number*> parameter indicates which entry's remark will be replaced.

The **remark** <*comment-text*> adds a comment to the ACL entry. The remark can have up to 128 characters in length. The comment must be entered separately from the actual ACL entry; that is, you cannot enter the ACL entry and the ACL comment with the same command. Also, in order for the remark to be displayed correctly in the output of **show** commands, the comment must be entered immediately before the ACL entry it describes.

Complete the syntax by specifying any options you want for the ACL entry.

Numbered ACLs: deleting a comment

To delete a remark from a named ACL entry, enter the following command.

```
ServerIronADX(config)# access-list 99 delete 2 remark
```

Syntax: **delete** <*line-number*> **remark**

Named ACLs: adding a comment to a new ACL

You can add, insert, replace, and delete ACL entry remarks. To add a comment, do the following.

1. Use the **show access-list** command to display the contents of the ACL. For example, you may have an ACL named "melon" and a **show access-list** command shows that it has only one entry.

```
ServerIronADX(config)#show access-list melon
Standard IP access-list 99
deny host 1.2.4.5
```

2. Add a new entry with a remark to this named ACL by entering commands such as the following.

```
ServerIronADX(config)# ip access-list standard melon
ServerIronADX(config-std-nacl)# remark Deny traffic from Marketing
ServerIronADX(config-std-nacl)# deny 5.6.7.8
```

3. Enter a **show access-list** command displays the new ACL entry with its remark.

```
ServerIronADX(config)# show access-list melon
Standard IP access-list melon
deny host 1.2.4.5
   Deny traffic from Marketing
permit host 5.6.7.8
```

Syntax: **ip access-list standard | extended** <*acl-name*> **|** <*acl-num*>

Syntax: **remark** <*comment-text*>

Syntax: **deny** <*options*> | **permit** <*options*>

The **standard** | **extended** parameter indicates the ACL type.

The <*acl-name*> parameter is the ACL name. You can specify a string of up to 256 alphanumeric characters. You can use blanks in the ACL name if you enclose the name in quotation marks (for example, “ACL for Net1”). The <*acl-num*> parameter allows you to specify an ACL number if you prefer. If you specify a number, enter a number from 1 – 99 for standard ACLs or 100 – 199 for extended ACLs.

The **remark** <*comment-text*> adds a comment to the ACL entry that you are about to create. The comment can have up to 128 characters in length. The comment must be entered separately from the actual ACL entry; that is, you cannot enter the ACL entry and the ACL comment with the same command. Also, in order for the remark to be displayed correctly in the output of show commands, the comment must be entered immediately before the ACL entry it describes.

Enter **deny** to deny the specified traffic or **permit** to allow the specified traffic. Complete the configuration by specifying <*options*> for the standard or extended ACL entry.

Named ACLs: inserting or replacing comments to existing ACL entries

To insert a comment to an existing entry in the ACL named melon, or to replace a comment for an ACL entry, display the list of entries in the ACL.

```
ServerIronADX(config)# show access-list melon
Standard IP access-list melon
deny host 1.2.4.5
permit host 5.6.7.8
permit any
```

To add the comment "Permit all users" to the second entry in the list, enter a command such as the following.

```
ServerIronADX(config)# ip access-list standard melon
ServerIronADX(config-std-nacl)# insert 3 remark Permit all users
```

Use the **show access-list** command to display the updated ACL.

```
ServerIronADX(config)#show access-list melon
Standard IP access-list melon
deny host 1.2.4.5
permit host 5.6.7.8
   Permit all users
permit ip any any
```

To replace the comment for the third entry, enter commands such as the following.

```
ServerIronADX(config)# ip access-list standard melon
ServerIronADX(config-std-nacl)# replace 3 remark All users allowed
```

Entering the **show access-list** command displays the updated ACL.

```
ServerIronADX(config)# show access-list melon
Standard IP access-list melon
deny host 1.2.4.5
permit host 5.6.7.8
   All users allowed
permit ip any any
```

Syntax: **ip access-list standard | extended** <*acl-name*> | <*acl-num*>

Syntax: **insert** <*line-number*> | **replace** <*line-number*> **remark** <*comment-text*>

The **standard** | **extended** parameter indicates the ACL type.

The <*acl-name*> parameter is the ACL name. You can specify a string of up to 256 alphanumeric characters. You can use blanks in the ACL name if you enclose the name in quotation marks (for example, "ACL for Net1"). The <*acl-num*> parameter allows you to specify an ACL number if you prefer. You can specify a number from 1 – 99 for standard ACLs or 100 – 199 for extended ACLs.

The **insert** <*line-number*> parameter indicates into which entry the comment is to be added. The **replace** <*line-number*> command indicates which remarks will be replaced.

The **remark** <*comment-text*> adds a comment to the ACL entry. The remark can have up to 128 characters in length. The comment must be entered separately from the actual ACL entry; that is, you cannot enter the ACL entry and the ACL comment with the same **access-list** command. Also, in order for the remark to be displayed correctly in the output of **show** commands, the comment must be entered immediately before the ACL entry it describes.

Complete the syntax by specifying options for the ACL entry.

Deleting a remark from a named ACL

To delete a remark from a named ACL, enter the following command.

```
ServerIronADX(config)# ip access-list standard melon
ServerIronADX(config-std-nacl)# delete 3 remark
```

Syntax: delete <*line-number*> **remark**

Displaying a list of ACL entries

The **show access-list** and **show ip access-list** commands displays ACL entries with line numbers.

Numbered ACLs

To display the contents of numbered ACLs, enter a command such as the following.

```
ServerIronADX# show access-list 99
Standard IP access list 99
deny host 1.2.4.5
deny host 5.6.7.8
permit any
```

Syntax: show access-list <*acl-num*> **| all**

Named ACLs

To display the contents of named ACLs, enter a command such as the following.

```
ServerIronADX# show ip access-list melon
Standard IP access list melon
deny host 1.2.4.5
deny host 5.6.7.8
permit any
```

Syntax: show ip access-list <*acl-num*> **|** <*acl-name*>

Applying an ACLs to interfaces

Configuration examples in the section "Configuring numbered and named ACLs" on page 46 show that you apply ACLs to interfaces using the **ip access-group** command. This section present additional information about applying ACLs to interfaces.

Reapplying modified ACLs

If you make an ACL configuration change, you must reapply the ACLs to their interfaces to place the change into effect.

An ACL configuration change includes any of the following:

- Adding, changing, or removing an ACL or an entry in an ACL
- Changing a PBR policy

To reapply ACLs following an ACL configuration change, enter the following command at the global CONFIG level of the CLI.

```
ServerIronADX(config)# ip rebind-acl all
```

Syntax: [no] ip rebind-acl <*num*> **|** <*name*> **| all**

ACL logging

You may want the software to log entries for ACLs in the syslog. This section present the how logging is processed by rule-based ACLs.

Rule-based ACLs do not support the **log** option. Even when rule-based ACLs are enabled, if an ACL entry has the **log** option, traffic that matches that ACL is sent to the CPU for processing. Depending on how many entries have the log option and how often packets match those entries, ACL performance can be affected.

If your configuration already contains ACLs that you want to use with rule-based ACLs, but some of the ACLs contain the **log** option, the software changes the ACL mode to flow-based for the traffic flows that match the ACL. Changing the mode to flow-based enables the device to send the matching flows to the CPU for processing. This is required because the CPU is needed to generate the Syslog message.

You can globally disable ACL logging without the need to remove the **log** option from each ACL entry. When you globally disable ACL logging, the ACL entries remain unchanged but the **log** option is ignored and the ACL can use the rule-based ACL mode. This enables you to use the ACLs in the rule-based ACL mode. You also can configure the device to copy traffic that is denied by a rule-based ACL to an interface. This option allows you to monitor the denied traffic without sending the traffic to the CPU.

To globally disable ACL logging, enter the following command at the global CONFIG level of the CLI.

```
ServerIronADX(config)# ip access-list disable-log-to-cpu
```

Syntax: [no] ip access-list disable-log-to-cpu

To re-enable ACL logging, enter the following command.

```
ServerIronADX(config)# no ip access-list disable-log-to-cpu
```

Syslog message for changed ACL mode

If the device changes the ACL mode from rule-based to flow-based, the device generates one of the following Syslog notification messages:

- ACL insufficient L4 session resource, using flow based ACL instead.
- ACL exceed max DMA L4 cam resource, using flow based ACL instead. Refer to "Specifying the maximum number of CAM entries for rule-based ACLs" on page 45.
- ACL insufficient L4 cam resource, using flow based ACL instead.

Copying denied traffic to a mirror port for monitoring

Although rule-based ACLs do not support ACL logging, you nonetheless can monitor the traffic denied by rule-based ACLs. To do so, attach a protocol analyzer to a port and enable the device to redirect traffic denied by ACLs to that port.

To redirect traffic denied by ACLs, enter the following command at the interface configuration level.

```
ServerIronADX(config-if-1/1)# ip access-group redirect-deny-to-interf
```

Syntax: [no] ip access-group redirect-deny-to-interf

Enter the command on the port to which you want the denied traffic to be copied.

NOTE
The software requires that an ACL has already been applied to the interface.

When you enable redirection, the deny action of the ACL entry is still honored. Traffic that matches the ACL is not forwarded.

Displaying ACL log entries

The first time an entry in an ACL permits or denies a packet and logging is enabled for that entry, the software generates a Syslog message and an SNMP trap. Messages for packets permitted or denied by ACLs are at the warning level of the Syslog.

When the first Syslog entry for a packet permitted or denied by an ACL is generated, the software starts an ACL timer. After this, the software sends Syslog messages every one to ten minutes, depending on the value of the timer interval. If an ACL entry does not permit or deny any packets during the timer interval, the software does not generate a Syslog entry for that ACL entry.

NOTE
For an ACL entry to be eligible to generate a Syslog entry for permitted or denied packets, logging must be enabled for the entry. The Syslog contains entries only for the ACL entries that deny packets and have logging enabled.

To display Syslog entries, enter the following command from any CLI prompt.

```
ServerIronADX(config)# show log
Syslog logging: enabled (0 messages dropped, 0 flushes, 0 overruns)
    Buffer logging: level ACDMEINW, 38 messages logged
    level code: A=alert C=critical D=debugging M=emergency E=error
                I=informational N=notification W=warning

Log Buffer (50 entries):

21d07h02m40s:warning:list 101 denied tcp 209.157.22.191(0)(Ethernet 4/18
0010.5a1f.77ed) -> 198.99.4.69(http), 1 event(s)
00d07h03m30s:warning:list 101 denied tcp 209.157.22.26(0)(Ethernet 4/18
0010.5a1f.77ed) -> 198.99.4.69(http), 1 event(s)
00d06h58m30s:warning:list 101 denied tcp 209.157.22.198(0)(Ethernet 4/18
0010.5a1f.77ed) -> 198.99.4.69(http), 1 event(s)
```

In this example, the two-line message at the bottom is the first entry, which the software immediately generates the first time an ACL entry permits or denies a packet. In this case, an entry in ACL 101 denied a packet. The packet was a TCP packet from host 209.157.22.198 and was destined for TCP port 80 (HTTP) on host 198.99.4.69.

When the software places the first entry in the log, the software also starts the five-minute timer for subsequent log entries. Thus, five minutes after the first log entry, the software generates another log entry and SNMP trap for denied packets.

In this example, the software generates the second log entry five minutes later.

The time stamp for the third entry is much later than the time stamps for the first two entries. In this case, no ACLs denied packets for a very long time. In fact, since no ACLs denied packets during the five-minute interval following the second entry, the software stopped the ACL log timer. The software generated the third entry as soon as the ACL denied a packet. The software restarted the five-minute ACL log timer at the same time. As long as at least one ACL entry permits or denies a packet, the timer continues to generate new log entries and SNMP traps every five minutes.

You can also configure the maximum number of ACL-related log entries that can be added to the system log over a one-minute period. For example, to limit the device to 100 ACL-related syslog entries per minute.

```
ServerIronADX(config)# max-acl-log-num 100
```

Syntax: [no] max-acl-log-num <*num*>

You can specify a number between 0 – 4096. The default is 256. Specifying 0 disables all ACL logging.

Displaying ACL statistics for flow-based ACLs

To display ACL statistics for flow-based ACLs, enter the following command.

```
ServerIronADX(config)# show ip acl-traffic

ICMP inbound packets received 400
ICMP inbound packets permitted 200
ICMP inbound packets denied 200
```

Syntax: show ip acl-traffic

The command lists a separate set of statistics for each of the following IP protocols:

- ICMP
- IGMP
- IGRP
- IP
- OSPF
- TCP
- UDP
- Protocol number, if an ACL is configured for a protocol not listed above

For TCP and UDP, a separate set of statistics is listed for each application port.

Clearing flow-based ACL statistics

To clear the ACL statistics, enter the following command at the Privileged EXEC level of the CLI.

```
ServerIronADX(config)# clear ip acl-traffic
```

Syntax: clear ip acl-traffic

Dropping all fragments that exactly match a flow-based ACL

For a packet fragment that is sent to the CPU for processing, the device compares the fragment's source and destination IP addresses against the interface's ACL entries. By default, if the fragment's source and destination IP addresses exactly match an ACL entry that also has Layer 4 information (source and destination TCP or UDP application ports), the device permits or denies the fragment according to the ACL.

On an individual interface basis, you can configure an IronCore device to automatically drop a fragment whose source and destination IP addresses exactly match an ACL entry that has Layer 4 information, even if that ACL entry's action is permit. To do so, enter the following command at the configuration level for an interface.

```
ServerIronADX(config-if-1/1)# ip access-group frag deny
```

Syntax: [no] ip access-group frag deny

Clearing the ACL statistics

Statistics on the ACL account report can be cleared:

- When a software reload occurs
- When the ACL is bound to or unbound from an interface
- When you enter the **clear access-list** command, as in the following example.

```
ServerIronADX(config)# clear access-list all
```

Syntax: clear access-list all | ethernet <*slot*>/<*port*>

Enter **all** to clear all statistics for all ACLs.

Use **ethernet** <*slot*>/<*port*> to clear statistics for ACLs a physical port.

Enabling ACL filtering of fragmented packets

Filtering fragmented packets for rule-based ACLs

By default, when a rule-based ACL is applied to a port, the port will use the ACL to permit or deny the first fragment of a fragmented packet, but forward subsequent fragments of the same packet in hardware. Generally, denying the first fragment of a packet is sufficient, since a transaction cannot be completed without the entire packet.

NOTE
The fragmentation support described in this section applies only to rule-based ACLs.

NOTE
Enhanced fragment handling is not supported on 10 Gigabit Ethernet modules. By default, 10 Gigabit Ethernet modules also forward the first fragment instead of using the ACLs to permit or deny the fragment.

For tighter control, you can enable CPU filtering of all packet fragments on a port. When you enable CPU filtering, the port sends all the fragments of a fragmented packet to the CPU. The CPU then permits or denies each fragment according to the ACL applied to the port. You can enable CPU filtering of fragments on individual ports.

You also can configure the port to drop all packet fragments.

To enable CPU filtering of packet fragments on an individual port, enter commands such as the following.

```
ServerIronADX(config)# interface ethernet 1/1
ServerIronADX(config-if-1/1)# ip access-group frag inspect
```

Syntax: [no] ip access-group frag inspect | deny

The **inspect | deny** parameter specifies whether you want fragments to be sent to the CPU or dropped:

- **inspect** – This option sends all fragments to the CPU.
- **deny** – This option begins dropping all fragments received by the port as soon as you enter the command. This option is especially useful if the port is receiving an unusually high rate of fragments, which can indicate a hacker attack.

Throttling the fragment rate

By default, when you enable CPU filtering of packet fragments, all fragments are sent to the CPU. Normally, the fragment rate in a typical network does not place enough additional load on the CPU to adversely affect performance. However, performance can be affected if the device receives a very high rate of fragments. For example, a misconfigured server or a hacker can affect the device's performance by flooding the CPU with fragments.

You can protect against fragment flooding by specifying the maximum number of fragments the device or an individual interface is allowed to send to the CPU in a one-second interval. If the device or an interface receives more than the specified number of fragments in a one-second interval, the device either drops or forwards subsequent fragments in hardware, depending on the action you specify. In addition, the device starts a holddown timer and continues to either drop or forward fragments until the holddown time expires.

The device also generates a Syslog message.

To specify the maximum fragment rate per second, enter commands such as the following.

```
ServerIronADX(config)# ip access-list frag-rate-on-system 15000 exceed-action
drop reset-interval 10
ServerIronADX(config)#ip access-list frag-rate-on-interface 5000 exceed-action
forward reset-interval 5
```

The first command sets the fragment threshold at 15,000 per second, for the entire device. If the device receives more than 15,000 packet fragments in a one-second interval, the device takes the specified action. The action specified with this command is to drop the excess fragments and continue dropping fragments for a holddown time of ten minutes. After the ten minutes have passed, the device starts sending fragments to the CPU again for processing.

The second command sets the fragment threshold at 5,000 for individual interfaces. If any interface on the device receives more than 5,000 fragments in a one-second interval, the device takes the specified action. In this case, the action is to forward the fragments in hardware without filtering them. The device continues forwarding fragments in hardware for five minutes before beginning to send fragments to the CPU again.

Both thresholds apply to the entire device. Thus, if an individual interface's fragment threshold is exceeded, the drop or forward action and the holddown time apply to all fragments received by the device.

Syntax: [no] ip access-list frag-rate-on-system <*num*> **exceed-action drop | forward reset-interval** <*mins*>

and

Syntax: [no] ip access-list frag-rate-on-interface <*num*> **exceed-action drop | forward reset-interval** <*mins*>

The <*num*> parameter specifies the maximum number of fragments the device or an individual interface can receive and send to the CPU in a one-second interval.

- **frag-rate-on-system** – Sets the threshold for the entire device. The device can send to the CPU only the number of fragments you specify per second, regardless of which interfaces the fragments come in on. If the threshold is exceeded, the device takes the exceed action you specify.
- **frag-rate-on-interface** – Sets the threshold for individual interfaces. If an individual interface receives more than the specified maximum number of fragments, the device takes the exceed action you specify.

The <*num*> parameter specifies the maximum number of fragments per second.

- For **frag-rate-on-system**, you can specify from 600 – 12800. The default is 6400.
- For **frag-rate-on-interface**, you can specify from 300 – 8000. The default is 4000.

The **drop | forward** parameter specifies the action to take if the threshold (<*num*> parameter) is exceeded:

- **drop** – fragments are dropped without filtering by the ACLs
- **forward** – fragments are forwarded in hardware without filtering by the ACLs

The <*mins*> parameter specifies the number of minutes the device will enforce the drop or forward action after a threshold has been exceeded. You can specify from 1 – 30 minutes, for **frag-rate-on-system** or **frag-rate-on-interface**.

Syslog messages for exceeded fragment thresholds

If a fragment threshold is exceeded, the device generates one of the following Syslog messages.

TABLE 11 Syslog messages for exceeded fragment threshold

Message level	Message	Explanation
Notification	ACL system fragment packet inspect rate <*rate*> exceeded	The <*rate*> indicates the maximum rate allowed.
Notification	ACL port fragment packet inspect rate <*rate*> exceeded on port <*portnum*>	The <*rate*> indicates the maximum rate allowed. The <*portnum*> indicates the port.

Enabling hardware filtering for packets denied by flow-based ACLs

By default, packets denied by ACLs are filtered by the CPU. You can enable the device to create CAM entries for packets denied by ACLs. This causes the filtering to occur in hardware instead of in the CPU.

When you enable hardware filtering of denied packets, the first time the device filters a packet denied by an ACL, the device sends the packet to the CPU for processing. The CPU also creates a CAM entry for the denied packet. Subsequent packets with the same address information are filtered using the CAM entry. The CAM entry ages out after two minutes if not used.

To enable hardware filtering of denied packets, enter the following command at the global CONFIG level of the CLI.

```
ServerIronADX(config)# hw-drop-acl-denied-packet
```

Syntax: [no] hw-drop-acl-denied-packet

Enabling strict TCP or UDP mode for flow-based ACLs

By default, when you use ACLs to filter TCP or UDP traffic, the Brocade device does not compare all TCP or UDP packets against the ACLs.

For TCP and UDP, the device first compares the source and destination information in a TCP control packet or a UDP packet against entries in the session table. The session table contains forwarding entries based on Layer 3 and Layer 4 information:

- If the session table contains a matching entry, the device forwards the packet, assuming that the first packet the device received with the same address information was permitted by the ACLs.
- If the session table does not contain a matching entry, the device sends the packet to the CPU, where the software compares the packet against the ACLs. If the ACLs permit the packet (explicitly by a permit ACL entry or implicitly by the absence of a deny ACL entry), the CPU creates a session table entry for the packet's forwarding information and forwards the packet.

For TCP, this behavior by default applies only to control packets, not to data packets. Control packets include packet types such as SYN (Synchronization) packets, FIN (Finish) packets, and RST (Reset) packets.

For tighter access or forwarding control, you can enable the device to perform strict TCP or UDP ACL processing. The following sections describe the strict modes in more detail.

Enabling strict TCP mode

By default, when you use ACLs to filter TCP traffic, the Brocade device does not compare all TCP packets against the ACLs. Instead, the device compares TCP control packets against the ACLs, but not data packets. Control packets include packet types such as SYN (Synchronization) packets, FIN (Finish) packets, and RST (Reset) packets.

In normal TCP operation, TCP data packets are present only if a TCP control session for the packets also is established. For example, data packets for a session never occur if the TCP SYN for that session is dropped. Therefore, by filtering the control packets, the Brocade device also implicitly filters the data packets associated with the control packets. This mode of filtering optimizes forwarding performance for TCP traffic by forwarding data packets without examining them. Since the data packets are present in normal TCP traffic only if a corresponding TCP control session is established, comparing the packets for the control session to the ACLs is sufficient for filtering the entire session including the data.

However, it is possible to generate TCP data packets without corresponding control packets, in test or research situations for example. In this case, the default ACL mode does not filter the data packets, since there is no corresponding control session to filter. To filter this type of TCP traffic, use the strict ACL TCP mode. This mode compares all TCP packets to the configured ACLs, regardless of whether the packets are control packets or data packets. If the ACLs permit the packet, the device creates a session entry for forwarding other TCP packets with the same Layer 3 and Layer 4 addresses.

NOTE
Regardless of whether the strict mode is enabled or disabled, the device always compares TCP control packets against the configured ACLs before creating a session entry for forwarding the traffic.

NOTE
If the device's configuration currently has ACLs associated with interfaces, remove the ACLs from the interfaces before changing the ACL mode.

To enable the strict ACL TCP mode, enter the following command at the global CONFIG level of the CLI.

```
ServerIronADX(config)# ip strict-acl-tcp
```

Syntax: [no] ip strict-acl-tcp

This command configures the device to compare all TCP packets against the configured ACLs before forwarding them.

To disable the strict ACL mode and return to the default ACL behavior, enter the following command.

```
ServerIronADX(config)# no ip strict-acl-tcp
```

NOTE
Enter the **ip rebind-acl** command at the global CONFIG level of the CLI to place the **ip strict-acl-tcp** or **no ip strict-acl-tcp** command into effect.

Enabling strict UDP mode

By default, when you use ACLs to filter UDP traffic, the Brocade device does not compare all UDP packets against the ACLs. Instead, the device compares the source and destination information against entries in the session table. The session table contains forwarding entries based on Layer 3 and Layer 4 information:

- If the session table contains a matching entry, the device forwards the packet, assuming that the first packet the device received that contains the same address information was permitted by the ACLs.
- If the session table does not contain a matching entry, the device sends the packet to the CPU, where the software compares the packet against the ACLs. If the ACLs permit the packet (explicitly by a permit ACL entry or implicitly by the absence of a deny ACL entry), the CPU creates a session table entry for the packet's forwarding information and forwards the packet.

For tighter control, the software provides the strict ACL UDP mode. When you enable strict UDP processing, the device sends every UDP packet to the CPU and compares the packet against the configured ACLs.

NOTE
If the device's configuration currently has ACLs associated with interfaces, remove the ACLs from the interfaces before changing the ACL mode.

To enable the strict ACL UDP mode, enter the following command at the global CONFIG level of the CLI.

```
ServerIronADX(config)# ip strict-acl-udp
```

Syntax: [no] ip strict-acl-udp

This command configures the device to compare all UDP packets against the configured ACLs before forwarding them.

To disable the strict ACL mode and return to the default ACL behavior, enter the following command.

```
ServerIronADX(config)# no ip strict-acl-udp
```

NOTE
Enter the **ip rebind-acl** command at the global CONFIG level of the CLI to place the **ip strict-acl-udp** or **no ip strict-acl-udp** command into effect.

Configuring ACL packet and flow counters

You can configure counters for packets and flows that match entries in an ACL. Using the CLI, you can display the contents of the counters and clear them:

- The ACL packet counter feature provides an accurate count of packets matching individual ACL entries.
- The ACL flow counter feature provides an approximate count of flows matching individual ACL entries. This feature can be used for troubleshooting purposes to provide an indication of flow activity against an ACL. Each time the Brocade device receives the first packet of a flow matching an entry in an ACL list, the flow counter for that ACL entry is incremented by one. If a flow lasts longer than two minutes, the flow counter for the ACL entry is incremented again.

  NOTE
  The ACL flow counter feature is designed to monitor the general volume of flow activity for an ACL. It is not intended to be used for accounting purposes.

The ACL flow and packet counters are incremented differently depending on whether packets are handled by the Management Processor (MP), and whether they are permit or deny flows.

The Management Processor (MP) handles flows as follows.

For flows handled by the Management Processor:

- For permit flows, only flows are counted. If a permit flow lasts longer than two minutes, the flow counter is incremented again.
- For deny flows, only packets are counted.

By default the ACL packet and flow counters are disabled. To activate them, enter the following command.

```
ServerIronADX(config)# enable-acl-counter
```

Syntax: [no] enable-acl-counter

Once the ACL packet and flow counters are enabled, you can disable them with the **no** form of the **enable-acl-counter** command. Disabling and then re-enabling the ACL packet and flow counters resets them to zero.

To display the packet and flow counters for ACL 100.

```
ServerIronADX# show access-list 100
Extended IP access list 100 (Total flows: 432, Total packets: 42000)
  permit tcp 1.1.1.0 0.0.0.255 any (Flows: 80, Packets: 12900)
  deny udp 1.1.1.0 0.0.0.255 any (Flows: 121, Packets: 20100)
  permit ip 2.2.2.0 0.0.0.255 any (Flows: 231, Packets: 9000)
```

Syntax: show access-list <*acl-num*> **|** <*acl-name*> **| all**

To clear the flow counters for ACL 100.

```
ServerIronADX# clear access-list 100
```

Syntax: clear access-list <*acl-num*> **|** <*acl-name*> **| all**

ACLs and ICMP

This section describes how ACLs can be used to filter traffic based on ICMP packets.

Using flow-based ACLs to filter ICMP packets based on the IP packet length

To configure an extended ACL that filters based on the IP packet length of ICMP packets, enter commands such as the following.

```
ServerIronADX(config)#access-list 105 deny icmp any any echo ip-pkt-len 92
ServerIronADX(config)#access-list 105 deny icmp any any echo ip-pkt-len 100
ServerIronADX(config)#access-list 105 permit ip any any
```

The commands in this example deny (drop) ICMP echo request packets that contain a total length of 92 or 100 in the IP header field. You can specify an IP packet length of 1 – 65535. Refer to the section "ICMP filtering with flow-based ACLs" on page 74 for additional information on using ICMP to filter packets.

ICMP filtering with flow-based ACLs

Most Brocade software releases that support flow-based ACLs filter traffic based on the following ICMP message types:

- echo
- echo-reply
- information-request
- mask-reply
- mask-request
- parameter-problem
- redirect
- source-quench
- time-exceeded
- timestamp-reply
- timestamp-request
- unreachable

- *<num>*

Also, to create ACL policies that filter ICMP message types, you can either enter the description of the message type or enter its type and code IDs. Furthermore ICMP message type filtering is now available for rule-based ACLs on BigIron Layer 2 Switch and Layer 3 Switch images.

Numbered ACLs

For example, to deny the echo message type in a numbered ACL, enter commands such as the following when configuring a numbered ACL.

```
ServerIronADX(config)# access-list 109 deny ICMP any any echo
```

or

```
ServerIronADX(config)# access-list 109 deny ICMP any any 8 0
```

Syntax: **[no] access-list** *<num>*

Syntax: **deny | permit icmp** *<source-ip-address>* **|** *<source-ip-address/subnet-mask>* **| any | host** *<source-host>*
<destination-ip-address> **|** *<destination-ip-address/subnet-mask>* **| any | host** *<destination-host>*
<icmp-type> **|** *<icmp-type-number>* *<icmp-code-number>*

The **deny | permit** parameter indicates whether packets that match the policy are dropped or forwarded.

You can either enter the name of the message type for *<icmp-type>* or the type number and code number of the message type. Refer to Table 12 on page 76 for valid values.

Named ACLs

For example, to deny the administratively-prohibited message type in a named ACL, enter commands such as the following.

```
ServerIronADX(config)# ip access-list extended melon
ServerIronADX(config-ext-nacl)# deny ICMP any any administratively-prohibited
```

or

```
ServerIronADX(config)# ip access-list extended melon
ServerIronADX(config-ext-nacl)# deny ICMP any any 3 13
```

Syntax: **[no] ip access-list extended** *<acl-num>* **|** *<acl-name>*

Syntax: **deny | permit icmp** *<source-ip-address>* **|** *<source-ip-address/subnet-mask>* **| any | host** *<source-host>*
<destination-ip-address> **| destination-ip-address/subnet-mask> | any | host** *<destination-host>*
<icmp-type> **|** *<icmp-type-number>* *<icmp-code-number>*

The **extended** parameter indicates the ACL entry is an extended ACL.

The *<acl-name>* | *<acl-num>* parameter allows you to specify an ACL name or number. If using a name, specify a string of up to 256 alphanumeric characters. You can use blanks in the ACL name if you enclose the name in quotation marks (for example, “ACL for Net1”). The *<acl-num>* parameter allows you to specify an ACL number if you prefer. If you specify a number, enter a number from 100 – 199 for extended ACLs.

The **deny | permit** parameter indicates whether packets that match the policy are dropped or forwarded.

You can either use the *<icmp-type>* and enter the name of the message type or use the *<icmp-type-number>* *<icmp-ode-number>* parameter and enter the type number and code number of the message. Refer to Table 12 for valid values.

NOTE
"X" in the Type-Number or Code-Number column in Table 12 means the device filters any traffic of that ICMP message type.

TABLE 12 ICMP message types and codes

ICMP message type	Type	Code
administratively-prohibited	3	13
any-icmp-type	x	x
destination-host-prohibited	3	10
destination-host-unknown	3	7
destination-net-prohibited	3	9
destination-network-unknown	3	6
echo	8	0
echo-reply	0	0
general-parameter-problem **NOTE:** This message type indicates that required option is missing.	12	1
host-precedence-violation	3	14
host-redirect	5	1
host-tos-redirect	5	3
host-tos-unreachable	3	12
host-unreachable	3	1
information-request	15	0
log		
mask-reply	18	0
mask-request	17	0
net-redirect	5	0
net-tos-redirect	5	2
net-tos-unreachable	3	11
net-unreachable	3	0
packet-too-big	3	4
parameter-problem **NOTE:** This message includes all parameter problems	12	0
port-unreachable	3	3
precedence-cutoff	3	15

TABLE 12 ICMP message types and codes (Continued)

ICMP message type	Type	Code
protocol-unreachable	3	2
reassembly-timeout	11	1
redirect **NOTE:** This includes all redirects.	5	x
router-advertisement	9	0
router-solicitation	10	0
source-host-isolated	3	8
source-quench	4	0
source-route-failed	3	5
time-exceeded	11	x
timestamp-reply	14	0
timestamp-request	13	0
ttl-exceeded	11	0
unreachable **NOTE:** This includes all unreachable messages	3	x

Using ACLs and NAT on the same interface (flow-based ACLs)

You can use ACLs and NAT on the same interface, as long as you follow these guidelines:

- You must use the **ip strict-acl-tcp** command when configuring ACLs and NAT is configured on the same Layer 2 Switch. (Refer to the instructions below on how to use this command.)
- Do not enable NAT on an interface until you have applied ACLs (as described below) to the interface. If NAT is already enabled, you must disable it, apply the ACLs, then re-enable NAT on the interface.
- Enable the strict TCP mode.
- On the inside NAT interface (the one connected to the private addresses), apply inbound ACLs that permit TCP, UDP, and ICMP traffic to enter the device from the private sub-net.

You can use a standard ACL to permit all traffic (including TCP, UDP, and ICMP traffic) or an extended ACL with separate entries to explicitly permit TCP, UDP, and ICMP traffic.

NOTE
You do not need to apply ACLs to permit TCP, UDP, and ICMP traffic unless you are applying other ACLs to the interface as well. If you do not plan to apply any ACLs to a NAT interface, then you do not need to apply the ACLs to permit TCP, UDP, and ICMP traffic.

Here is an example of how to configure device to use ACLs and NAT on the same interfaces. In this example, the inside NAT interface is port 1/1 and the outside NAT interface is port 2/2.

The following commands enable the strict TCP mode and configure an ACL to permit all traffic from the 10.10.200.x sub-net. A second ACL denies traffic from a specific host on the Internet.

```
ServerIronADX(config)# ip strict-acl-tcp
ServerIronADX(config)# access-list 1 permit 10.10.200.0 0.0.0.255
ServerIronADX(config)# access-list 2 deny 209.157.2.184
```

The following commands configure global NAT parameters.

```
ServerIronADX(config)# ip nat inside source list 1 pool outadds overload
ServerIronADX(config)# ip nat pool outadds 204.168.2.1 204.168.2.254 netmask
255.255.255.0
```

The following commands configure the inside and outside NAT interfaces. Notice that the ACLs are applied to the inbound direction on the inside NAT interface, and are applied ***before*** NAT is enabled. In this example, ACL 1 permits all traffic to come into the inside interface from the private sub-net. ACL 2 denies traffic from a specific host from going out the interface to the private sub-net.

```
ServerIronADX(config)# interface ethernet 1/1
ServerIronADX(config-if-1/1)# ip address 10.10.200.1 255.255.255.0
ServerIronADX(config-if-1/1)# ip access-group 1 in
ServerIronADX(config-if-1/1)# ip access-group 2 out
ServerIronADX(config-if-1/1)# ip nat inside
ServerIronADX(config-if-1/1)# interface ethernet 2/2
ServerIronADX(config-if-2/2)# ip address 204.168.2.78 255.255.255.0
ServerIronADX(config-if-2/2)# ip nat outside
```

NOTE
Enter the **ip rebind-acl** command at the global CONFIG level of the CLI to place the **ip strict-acl-tcp** command into effect.

Displaying ACL bindings

You can display which ACLs (IPv4 and IPv6) are bound to which interfaces as shown in the following.

```
ServerIronADX# show access-list bindings
Access-list binding configuration:
!
interface ethernet 2
ip access-group 2 in
ipv6 traffic-filter acl1 in
!
interface ve 2
ip access-group 111 in
ipv6 traffic-filter acl2 out
```

Syntax: show access-list bindings

Troubleshooting rule-based ACLs

Use the following methods to troubleshoot a rule-based ACL:

- To display the number of Layer 4 CAM entries being used by each ACL, enter the **show access-list** *<acl-num>* | *<acl-name>* | **all** command. Refer to "Displaying the number of Layer 4 CAM entries" on page 45.

- To view the types of packets being received on an interface, enable ACL statistics using the **enable-acl-counter** command, reapply the ACLs by entering the **ip rebind-acl all** command, then display the statistics by entering the **show ip acl-traffic** command.
- To determine whether an ACL entry is correctly matching packets, add the **log** option to the ACL entry, then reapply the ACL. This forces the device to send packets that match the ACL entry to the CPU for processing. The **log** option also generates a Syslog entry for packets that are permitted or denied by the ACL entry.
- To determine whether the issue is specific to fragmentation, remove the Layer 4 information (TCP or UDP application ports) from the ACL, then reapply the ACL.

If you are using another feature that requires ACLs, either use the same ACL entries for filtering and for the other feature, or change to flow-based ACLs.

Chapter

IPv6 Access Control Lists 5

ACL overview

ServerIron ADX supports IPv6 Access Control Lists (ACLs) in software. You can configure up to 100 IPv6 ACLs. By default, IPv6 ACLs are processed in hardware and all IPv6 ACL rules are stored in TCAM.

An IPv6 ACL is composed of one or more conditional statements that pose an action (permit or deny) if a packet matches a specified source or destination prefix. There can be up to 1024 IPv6 ACL statements per device. When the maximum number of IPv6 ACL rules are reached, the following error message will display on the console:

```
IPv6 Hardware ACL rules cannot be configured,exceeds the maximum hardware limit of
1024 entries
Insufficient hardware resource for binding the ACL scale1 to interface Port or
Slot/Port.
```

In ACLs with multiple statements, you can specify a priority for each statement.The specified priority determines the order in which the statement appears in the ACL. The last statement in each IPv6 ACL is an implicit deny statement for all packets that do not match the previous statements in the ACL.

You can configure an IPv6 ACL on a global basis, then apply it to the incoming IPv6 packets on specified interfaces. You can apply only one IPv6 ACL to an interface's incoming traffic. When an interface receives an IPv6 packet, it applies the statement within the ACL in their order of appearance to the packet. As soon as a match occurs, the ServerIron ADX takes the specified action (permit or deny the packet) and stops further comparison for that packet.

Brocade's IPv6 ACLs enable traffic filtering based on the following information:

- IPv6 protocol
- Source IPv6 address
- Destination IPv6 address
- IPv6 message type
- Source TCP or UDP port (if the IPv6 protocol is TCP or UDP)
- Destination TCP or UDP port (if the IPv6 protocol is TCP or UDP)

The IPv6 protocol can be one of the following well-known names or any IPv6 protocol number from 0 – 255:

- Authentication Header (AHP)
- Encapsulating Security Payload (ESP)
- Internet Control Message Protocol (ICMP)
- Internet Protocol Version 6 (IPv6)
- Stream Control Transmission Protocol (SCTP)
- Transmission Control Protocol (TCP)

- User Datagram Protocol (UDP)

NOTE
TCP and UDP filters will be matched only if they are listed as the first option in the extension header.

For TCP and UDP, you also can specify a comparison operator and port name or number. For example, you can configure a policy to block web access to a specific website by denying all TCP port 80 (HTTP) packets from a specified source IPv6 address to the website's IPv6 address.

This chapter contains the following sections:

- "Configuring an IPv6 ACL" on page 83
- "Applying an IPv6 ACL to an interface" on page 90
- "Displaying ACLs" on page 91

Configuration Notes

- Either IPv6 must be enabled globally or an IPV6 address must be configured on an interface before IPv6 ACLs can be configured.
- You can create up to 100 IPv6 ACLs and can include up to 1024 entries or statements.
- Only named ACLs are supported.
- Inbound ACLs are supported.
- If an IPv6 ACL has the implicit **deny** condition, make sure it also **permits** the IPv6 link-local address, in addition to the global unicast address. Otherwise, routing protocols such as OSPF will not work. To view the link-local address, use the **show ipv6 interface** command.
- You cannot disable IPv6 on an interface to which an ACL is bound. Attempting to do so will cause the system to return the following error message.

```
ServerIronADX(config-if-e1000-7)#no ipv6 enable
Error:  Port 7 has IPv6 ACL configured.  Cannot disable IPv6
```

To disable IPv6, first remove the ACL from the interface.

Processing of IPv6 ACLs

There are two ways that IPv6 ACLs are processed in Brocade devices: in software and in hardware. This processing differs depending on the software release that you are running. These differences are described in the following sections.

Prior to release 12.3.01

Prior to release 12.3.01, IPv6 ACLs were processed as described in the following:

For deny and permit actions:

All permit and deny packets are forwarded to the BPs and the BPs perform the ACL processing.

Beginning with release 12.3.01 and later

Beginning with release 12.3.01, IPv6 ACLs are processed as described in the following:

For deny actions:

All deny packets are dropped in hardware.

For permit actions:

For all traffic, packets are processed in hardware and then forwarded to the BPs. The BPs do not take any action on the ACLs.

Backwards compatibility option:

You can use the **ipv6 flow-based-acl-enable** command to provide backwards compatibility for IPv6 ACL processing. If this command is configured, packets are processed in hardware and then forwarded to the BPs where the BPs also process the ACLs. This command is configured as shown in the following.

```
ServerIronADX(config)# ipv6 flow-based-acl-enable
```

Syntax: ipv6 flow-based-acl-enable

Configuring an IPv6 ACL

To configure an IPv6 ACL, do the following:

1. Create the IPv6 ACL.
2. Apply the IPv6 ACL to the interface.

Example Configurations

To configure an access list that blocks all Telnet traffic received on port 1/1 from IPv6 host 2000:2382:e0bb::2, enter the following commands.

```
ServerIronADX(config)# ipv6 access-list fdry
ServerIronADX(config-ipv6-access-list-fdry)# deny tcp host 2000:2382:e0bb:
:2 any eq telnet
ServerIronADX(config-ipv6-access-list-fdry)# permit ipv6 any any
ServerIronADX(config-ipv6-access-list-fdry)# exit
ServerIronADX(config)# int eth 1/1
ServerIronADX(config-if-1/1)# ipv6 traffic-filter fdry in
ServerIronADX(config)# write memory
```

Here is another example of commands for configuring an ACL and applying it to an interface.

```
ServerIronADX(config)# ipv6 access-list netw
ServerIronADX(config-ipv6-access-list-netw)# permit icmp 2000:2383:
e0bb::/64 2001:3782::/64
ServerIronADX(config-ipv6-access-list-netw)# deny ipv6 host 2000:2383:
e0ac::2 host 2000:2383:e0aa:0::24
ServerIronADX(config-ipv6-access-list-netw)# deny udp any any
ServerIronADX(config-ipv6-access-list-netw)# permit ipv6 any any
```

The first condition permits ICMP traffic from hosts in the 2000:2383:e0bb::x network to hosts in the 2001:3782::x network.

The second condition denies all IPv6 traffic from host 2000:2383:e0ac::2 to host 2000:2383:e0aa:0::24.

The third condition denies all UDP traffic.

The fourth condition permits all packets that are not explicitly denied by the other entries. Without this entry, the ACL would deny all incoming IPv6 traffic on the ports to which you assigned the ACL.

The following commands apply the ACL "netw" to the incoming traffic on port 1/2 and to the incoming traffic on port 4/3.

```
ServerIronADX(config)# int eth 1/2
ServerIronADX(config-if-1/2)# ipv6 traffic-filter netw in
ServerIronADX(config-if-1/2)# exit
ServerIronADX(config)# int eth 4/3
ServerIronADX(config-if-4/3)# ipv6 traffic-filter netw in
ServerIronADX(config)# write memory
```

Here is another example:

```
ServerIronADX(config)# ipv6 access-list nextone
ServerIronADX(config-ipv6-access-list rtr)# deny tcp 2001:1570:21::/24
2001:1570:22::/24
ServerIronADX(config-ipv6-access-list rtr)# deny udp any range 5 6
2001:1570:22::/24
ServerIronADX(config-ipv6-access-list rtr)# permit ipv6 any any
ServerIronADX(config-ipv6-access-list rtr)# write memory
```

The first condition in this ACL denies TCP traffic from the 2001:1570:21::x network to the 2001:1570:22::x network.

The next condition denies UDP packets from any source with source UDP port in ranges 5 to 6 and whose destination is to the 2001:1570:22::/24 network.

The third condition permits all packets containing source and destination addresses that are not explicitly denied by the first two. Without this entry, the ACL would deny all incoming IPv6 traffic on the ports to which you assign the ACL.

A **show running-config** command displays the following:

```
ServerIronADX(config)# show running-config
ipv6 access-list rtr
deny tcp 2001:1570:21::/24 2001:1570:22::/24
deny udp any range 5 6 2001:1570:22::/24
permit ipv6 any any
```

A **show ipv6 access-list** command displays the following:

```
ServerIronADX(config)# sh ipv6 access-list rtr
ipv6 access-list rtr: 3 entries
10: deny tcp 2001:1570:21::/24 2001:1570:22::/24
20: deny udp any range 5 6 2001:1570:22::/24
30: permit ipv6 any any
```

The following commands apply the ACL “rtr” to the incoming traffic on ports 2/1 and 2/2.

```
ServerIronADX(config)# int eth 2/1
ServerIronADX(config-if-2/1)# ipv6 traffic-filter rtr in
ServerIronADX(config-if-2/1)# exit
ServerIronADX(config)# int eth 2/2
ServerIronADX(config-if-2/2)# ipv6 traffic-filter rtr in
ServerIronADX(config)# write memory
```

Default and Implicit IPv6 ACL Action

The default action when no IPv6 ACLs are configured on an interface is to permit all IPv6 traffic. However, once you configure an IPv6 ACL and apply it to an interface, the default action for that interface is to deny all IPv6 traffic that is not explicitly permitted on the interface.

- If you want to tightly control access, configure ACLs consisting of permit entries for the access you want to permit. The ACLs implicitly deny all other access.

- If you want to secure access in environments with many users, you might want to configure ACLs that consist of explicit deny entries, then add an entry to permit all access to the end of each ACL. The permit entry permits packets that are not denied by the deny entries.

Every IPv6 ACL has the following implicit condition as its last match conditions:

deny ipv6 any any – Denies IPv6 traffic. You must enter a **permit ipv6 any any** as the last statement in the access-list if you want to permit IPv6 traffic that were not denied by the previous statements.

NOTE
If an IPv6 ACL has the implicit **deny** condition, make sure it also **permits** the IPv6 link-local address, in addition to the global unicast address. Otherwise, routing protocols such as OSPF will not work. To view the link-local address, use the **show ipv6 interface** command.

The conditions are applied in the order shown above, with **deny ipv6 any any** as the last condition applied.

For example, if you want to deny ICMP neighbor discovery acknowledgement, then permit any remaining IPv6 traffic, enter commands such as the following:

```
ServerIronADX(config)# ipv6 access-list netw
ServerIronADX(config-ipv6-access-list-netw)# permit icmp 2000:2383:e0bb:
:/64 2001:3782::/64
ServerIronADX(config-ipv6-access-list-netw)# deny icmp any any nd-na
ServerIronADX(config-ipv6-access-list-netw)# permit ipv6 any any
```

The first permit statement permits ICMP traffic from hosts in the 2000:2383:e0bb::x network to hosts in the 2001:3782::x network.

The deny statement denies ICMP neighbor discovery acknowledgement.

The last entry permits all packets that are not explicitly denied by the other entries. Without this entry, the ACL will deny all incoming IPv6 traffic on the ports to which you assigned the ACL.

Furthermore, if you add the statement **deny icmp any any** in the access list, then all neighbor discovery messages will be denied. You must explicitly enter the **permit icmp any any nd-na** and **permit icmp any any nd-ns** statements just before the **deny icmp** statement if you want the ACLs to permit neighbor discovery as in the example below.

```
ServerIronADX(config)# ipv6 access-list netw
ServerIronADX(config-ipv6-access-list-netw)# permit icmp 2000:2383:e0bb:
:/64 2001:3782::/64
ServerIronADX(config-ipv6-access-list-netw)# permit icmp any any nd-na
ServerIronADX(config-ipv6-access-list-netw)# permit icmp any any nd-ns
ServerIronADX(config-ipv6-access-list-netw)# deny icmp any any
ServerIronADX(config-ipv6-access-list-netw)# permit ipv6 any any
```

ACL Syntax

NOTES: The following features are not supported:

- **ipv6-operator flow-label**
- **ipv6-operator fragments** when any protocol is specified. The option "fragments" can be specified only when "permit/deny ipv6" is specified. If you specify "tcp" or any other protocol instead of "ipv6" the keyword, "fragments" cannot be used.

- **ipv6-operator routing** when any protocol is specified. (Same limitation as for **ipv6-operator fragments**)
- The following **ICMPv6 Message Types** are not supported:

DECIMAL	<0-255> ICMP message type
beyond-scope	Destination Unreachable ICMP message, Beyond Scope
destination-unreachable	Destination Unreachable ICMP messages
dscp	Match dscp value in IPv6 packet
echo-reply	Echo Reply ICMP message
echo-request	Echo Request ICMP message
header	Parameter Problem ICMP Message,Header Error
hop-limit	Time Exceeded ICMP Message,In Transit
log	Log matches against this entry
mld-query	MLD Query Message
mld-reduction	MLD Reduction Message
mld-report	MLD Report Message
nd-na	ND Neighbor Advertisement Message
nd-ns	ND Neighbor Solicitation Message
next-header	Parameter Problem ICMP Message,Next Header
no-admin	Destination Unreachable ICMP Message, Administratively Prohibited
no-route	Destination Unreachable ICMP Message, No Route
packet-too-big	Packet Too Big ICMP Message
parameter-option	Parameter Problem ICMP Message,Option
parameter-problem	Parameter Problem ICMP Messages
port-unreachable	Destination Unreachable ICMP Message, Port Unreachable
reassembly-timeout	Time Exceeded ICMP Message, ReassemblyTimeout
renum-command	Renumber Command ICMP Message
renum-result	Renumber Result ICMP Message
renum-seq-number	Renumber Sequence Number ICMP Message
router-advertisement	Router Advertisement ICMP Message
router-renumbering	All Router Renumbering ICMP Messages
router-solicitation	Router Solicitation ICMP Message
sequence	Sequence number for this entry
time-exceeded	Time Exceeded ICMP Messages
traffic-policy	Attach traffic policy by name
unreachable	Destination Unreachable ICMP message,Address Unreachable

When creating ACLs, use the appropriate syntax below for the protocol you are filtering.

For IPv6 and Supported Protocols Other than ICMP, TCP, or UDP

Syntax: **[no] ipv6 access-list** *<acl-name>*

Syntax: **permit | deny** *<protocol>*
<ipv6-source-prefix/prefix-length> **| any | host** *<source-ipv6_address>*
<ipv6-destination-prefix/prefix-length> **| any | host** *<ipv6-destination-address>*
[ipv6-operator [*<value>*]] **[log]**

For ICMP

Syntax: **[no] ipv6 access-list** *<acl-name>*

Syntax: **permit | deny icmp** *<ipv6-source-prefix/prefix-length>* **| any | host** *<source-ipv6_address>*
<ipv6-destination-prefix/prefix-length> **| any | host** *<ipv6-destination-address>*
[ipv6-operator [*<value>*]]
[[*<icmp-type>*][*<icmp-code>*]] **|** [*<icmp-message>*] **[log]**

For TCP

Syntax: **[no] ipv6 access-list** *<acl-name>*

Syntax: **permit | deny** *<tcp>*
<ipv6-source-prefix/prefix-length> **| any | host** *<source-ipv6_address>* **[tcp-udp-operator [source-port-number]]**
<ipv6-destination-prefix/prefix-length> **| any | host** *<ipv6-destination-address>*
[tcp-udp-operator [destination-port- number]]
[ipv6-operator [<value>]] **[log]**

For UDP

Syntax: **[no] ipv6 access-list** *<acl-name>*

Syntax: **permit | deny** *<udp>*
<ipv6-source-prefix/prefix-length> **| any | host** *<source-ipv6_address>* **[tcp-udp-operator [source port number]]**
<ipv6-destination-prefix/prefix-length> **| any | host** *<ipv6-destination-address>*
[tcp-udp-operator [destination port number]]
[ipv6-operator [*<value>*]] **[log]**

TABLE 13 Syntax Descriptions

Arguments...	Description...
ipv6 access-list <acl-name>	Enables the IPv6 configuration level and defines the name of the IPv6 ACL. The <acl-name> can contain up to 199 characters and numbers, but cannot begin with a number and cannot contain any spaces or quotation marks.
permit	The ACL will permit (forward) packets that match a policy in the access list.
deny	The ACL will deny (drop) packets that match a policy in the access list.
icmp	Indicates the you are filtering ICMP packets.

TABLE 13 Syntax Descriptions

Arguments...	Description...
protocol	The type of IPv6 packet you are filtering. You can specify a well-known name for some protocols whose number is less than 255. For other protocols, you must enter the number. Enter “?” instead of a protocol to list the well-known names recognized by the CLI. IPv6 protocols include: AHP – Authentication Header ESP – Encapsulating Security Payload IPv6 – Internet Protocol version 6 SCTP – Stream Control Transmission Protocol
<ipv6-source-prefix>/<prefix-length>	The <ipv6-source-prefix>/<prefix-length> parameter specify a source prefix and prefix length that a packet must match for the specified action (deny or permit) to occur. You must specify the <ipv6-source-prefix> parameter in hexadecimal using 16-bit values between colons as documented in RFC 2373. You must specify the <prefix-length> parameter as a decimal value. A slash mark (/) must follow the <ipv6-prefix> parameter and precede the <prefix-length> parameter.
<ipv6-destination-prefix>/<prefix-length>	The <ipv6-destination-prefix>/<prefix-length> parameter specify a destination prefix and prefix length that a packet must match for the specified action (deny or permit) to occur. You must specify the <ipv6-destination-prefix> parameter in hexadecimal using 16-bit values between colons as documented in RFC 2373. You must specify the <prefix-length> parameter as a decimal value. A slash mark (/) must follow the <ipv6-prefix> parameter and precede the <prefix-length> parameter
any	When specified instead of the <ipv6-source-prefix>/<prefix-length> or <ipv6-destination-prefix>/<prefix-length> parameters, matches any IPv6 prefix and is equivalent to the IPv6 prefix::/0.
host	Allows you specify a host IPv6 address. When you use this parameter, you do not need to specify the prefix length. A prefix length of all128 is implied.
icmp-type	ICMP packets can be filtered by ICMP message type. The type is a number from 0 to 255.
icmp code	ICMP packets, which are filtered by ICMP message type can also be filtered by the ICMP message code. The code is a number from 0 to 255,
icmp-message	ICMP packets are filtered by ICMP messages. See the "Configuring IPv6 ICMP Features" section of the "Configuring IPv6 Connectivity" chapter of the *ServerIron ADX TrafficWorks Switching and Routing Guide*.
tcp	Indicates the you are filtering TCP packets.
udp	Indicates the you are filtering UDP packets.
<ipv6-source-prefix>/<prefix-length>	The <ipv6-source-prefix>/<prefix-length> parameter specify a source prefix and prefix length that a packet must match for the specified action (deny or permit) to occur. You must specify the <ipv6-source-prefix> parameter in hexadecimal using 16-bit values between colons as documented in RFC 2373. You must specify the <prefix-length> parameter as a decimal value. A slash mark (/) must follow the <ipv6-prefix> parameter and precede the <prefix-length> parameter.

TABLE 13 Syntax Descriptions

Arguments...	Description...
<ipv6-destination-prefix>/<prefix-length>	The <ipv6-destination-prefix>/<prefix-length> parameter specify a destination prefix and prefix length that a packet must match for the specified action (deny or permit) to occur. You must specify the <ipv6-destination-prefix> parameter in hexadecimal using 16-bit values between colons as documented in RFC 2373. You must specify the <prefix-length> parameter as a decimal value. A slash mark (/) must follow the <ipv6-prefix> parameter and precede the <prefix-length> parameter
any	When specified instead of the <ipv6-source-prefix>/<prefix-length> or <ipv6-destination-prefix>/<prefix-length> parameters, matches any IPv6 prefix and is equivalent to the IPv6 prefix::/0.
host	Allows you specify a host IPv6 address. When you use this parameter, you do not need to specify the prefix length. A prefix length of all128 is implied.
tcp-udp-operator	The <tcp-udp-operator> parameter can be one of the following: **eq** – The policy applies to the TCP or UDP port name or number you enter after **eq**. **gt** – The policy applies to TCP or UDP port numbers greater than the port number or the numeric equivalent of the port name you enter after **gt**. Enter "?" to list the port names. **lt** – The policy applies to TCP or UDP port numbers that are less than the port number or the numeric equivalent of the port name you enter after **lt**. **neq** – The policy applies to all TCP or UDP port numbers except the port number or port name you enter after **neq**. **range** – The policy applies to all TCP port numbers that are between the first TCP or UDP port name or number and the second one you enter following the **range** parameter. The range includes the port names or numbers you enter. For example, to apply the policy to all ports between and including 23 (Telnet) and 53 (DNS), enter the following: **range 23 53**. The first port number in the range must be lower than the last number in the range. The <source-port number> and <destination-port-number> for the tcp-udp-operator is the number of the port.
ipv6-operator	Allows you to filter the packets further by using one of the following options: • **dscp** – The policy applies to packets that match the traffic class value in the traffic class field of the IPv6 packet header. This operator allows you to filter traffic based on TOS or IP precedence. You can specify a value from 0 – 63. • **fragments** – The policy applies to fragmented packets that contain a non-zero fragment offset. **NOTE:** This option is not applicable to filtering based on source or destination port, TCP flags, and ICMP flags. • **routing** – The policy applies only to IPv6 source-routed packets. **NOTE:** This option is not applicable to filtering based on source or destination port, TCP flags, and ICMP flags.
log	Allows statistics that match the ACL statement to be entered in the Syslog.

ICMP Message Configurations

If you want to specify an ICMP message, you can enter one of the following:

- beyond-scope
- destination-unreachable
- echo-reply

- echo-request
- header
- hop-limit
- mld-query
- mld-reduction
- mld-report
- nd-na
- nd-ns
- next-header
- no-admin
- no-route
- packet-too-big
- parameter-option
- parameter-problem
- port-unreachable
- reassembly-timeout
- renum-command
- renum-result
- renum-seq-number
- router-advertisement
- router-renumbering
- router-solicitation
- sequence
- time-exceeded
- unreachable

NOTE
If you do not specify a message type, the ACL applies to all types ICMP messages types.

Applying an IPv6 ACL to an interface

To apply an IPv6 ACL to an interface, enter commands such as the following:

```
ServerIronADX(config)# interface ethernet 3/1
ServerIronADX(config-if-e100-3/1)# ipv6 traffic-filter access1 in
```

This example applies the IPv6 ACL “access1” to incoming IPv6 packets on Ethernet interface 3/1. As a result, Ethernet interface 3/1 denies all incoming packets from the site-local prefix fec0:0:0:2::/64 and the global prefix 2001:100:1::/48 and permits all other incoming packets.

Syntax: ipv6 traffic-filter <*ipv6-acl-name*> **in**

For the <*ipv6-acl-name*> parameter, specify the name of an IPv6 ACL created using the **ipv6 access-list** command.

The **in** keyword applies the specified IPv6 ACL to incoming IPv6 packets on the interface.

Displaying ACLs

To display the ACLs configured on a device, enter the **show ipv6 access-list** command. Here is an example:

```
ServerIronADX# show ipv6 access-list
ipv6 access-list v6-acl1: 1 entries
 deny ipv6 any any
ipv6 access-list v6-acl2: 1 entries
 permit ipv6 any any
ipv6 access-list v6-acl3: 2 entries
 deny ipv6 2001:aa:10::/64 any
 permit ipv6 any any
ipv6 access-list v6-acl4: 2 entries
 deny ipv6 2002:aa::/64 any
 permit ipv6 any any
ipv6 access-list v6-acl5: 6 entries
 permit tcp 2002:bb::/64 any
 permit ipv6 2002:bb::/64 any
 permit ipv6 2001:aa:101::/64 any
 permit ipv6 2001:aa:10::/64 2001:aa:102::/64
 permit ipv6 host 2001:aa:10::102 host 2001:aa:101::102
 permit ipv6 any any fragments
```

Syntax: show ipv6 access-list [<*access-list-name*>]

Displaying ACLs bound to an interface

To display ACLs bound to an interface, enter the **show access-list bindings** command. Here is an example:

```
ServerIronADX# show access-list bindings
Access-list binding configuration:
!
interface ethernet 1
ipv6 traffic-filter ipv61 in
!
interface ethernet 2
ipv6 traffic-filter icmp_any in
!
ServerIronADX 1000#
```

Syntax: show access-list bindings

Using an ACL to Restrict SSH Access

To configure an ACL that restricts SSH access to an IPv6 device, first create the named ACL with the ACL statements. Then use the **ssh access-group** command to restrich SSH access for IPv6:

```
ServerIronADX(config)# ipv6 access-list test2
ServerIronADX(config-ipv6-access-list test2)#  deny ipv6  host 2000:1::1 any log
ServerIronADX(config-ipv6-access-list test2)#  permit ipv6  2000:1::0/32 any
ServerIronADX(config-ipv6-access-list test2)#  permit ipv6  2000:2::0/32 any
ServerIronADX(config-ipv6-access-list test2)#  permit ipv6 host 2000:3::1 any
ServerIronADX(config-ipv6-access-list test2)# exit
ServerIronADX(config)# ssh access-group ipv6 test2
```

Syntax: [no] ssh access-group ipv6 <*acl-name*>

Using an ACL to Restrict Telnet Access

To configure an ACL that restricts Telnet access to an IPv6 device, first create the named ACL with the ACL statements. Then use the **telnet access-group** command to restrict Telnet access for IPv6:

```
ServerIronADX(config)# ipv6 access-list test1
ServerIronADX(config-ipv6-access-list test1)#  deny ipv6  host 2000:1::1 any log
ServerIronADX(config-ipv6-access-list test1)#  permit ipv6  2000:1::0/32 any
ServerIronADX(config-ipv6-access-list test1)#  permit ipv6  2000:2::0/32 any
ServerIronADX(config-ipv6-access-list test1)#  permit ipv6 host 2000:3::1 any
ServerIronADX(config-ipv6-access-list test1)# exit
ServerIronADX(config)# telnet access-group ipv6 test1
```

Syntax: telnet access-group ipv6 <*acl-name*>

Logging IPv6 ACLs

Logging for IPv6 ACLs is disabled by default. To enable logging, enable it for each IPv6 ACL, then include the logging option in an ACL statement. Logging at both levels need to be configured in order for statistics for packets that match the condition to be logged. For example:

```
ServerIronADX(config)# ipv6 access-list acl2
ServerIronADX(config-ipv6-access-list-acl2)# logging-enable
ServerIronADX(config-ipv6-access-list-acl2)# permit tcp host
2002:200:12d:1300:204:23ff:fec7:dabf any eq http
ServerIronADX(config-ipv6-access-list-acl2)# deny icmp 2002:200:12d:1300::/64 any
nd-na log
ServerIronADX(config-ipv6-access-list-acl2)# deny icmp 2002:200:12d:1300::/64 any
nd-ns log
ServerIronADX(config-ipv6-access-list-acl2)# deny icmp 2002:200:12d:1300::/64 any
echo-request log
ServerIronADX(config-ipv6-access-list-acl2)# deny icmp 2002:200:12d:1300::/64 any
echo-reply log
ServerIronADX(config-ipv6-access-list-acl2)# permit ipv6 any any
```

Syntax: [no] logging-enable

NOTE
Syntax for the **log** option in an IPv6 ACL statement are presented in the section "ACL Syntax" on page 85.

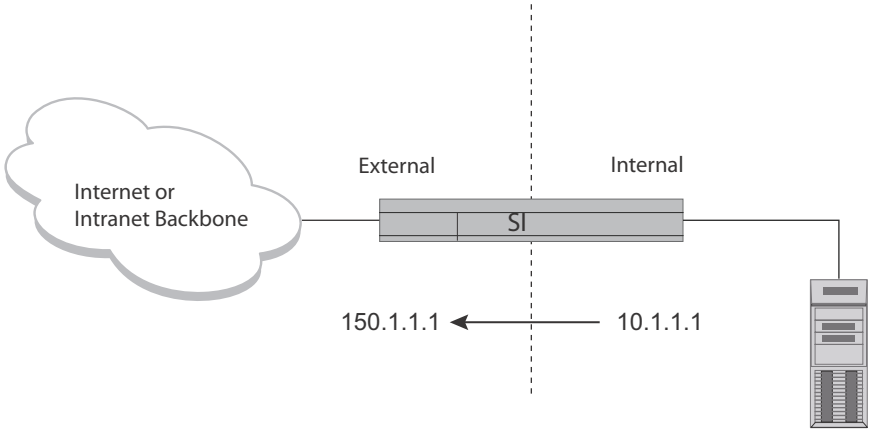
Chapter

6 Network Address Translation

Introduction

Network Address Translation (NAT) translates one IP address into another. For example, it translates an internal private IP address (nonregistered) into an external unique IP address (registered) used on the Internet.

FIGURE 15 Mapping an internal address to an external address



NAT also provides a more graceful renumbering strategy for organizations changing service providers or voluntarily renumbering into Classless Interdomain Routing (CIDR) blocks.

The standard NAT support described in this section provides translation for hosts attached to private networks on the ServerIron ADX, and is separate from the virtual IP address features provided for Server Load Balancing (SLB). For example, standard NAT is not related to source IP addresses used for multinetting the ServerIron ADX, performing health checks on remote servers, and so on.

Configuring NAT

The following types of NAT are supported:

- **Static NAT** — Maps a specific global IP address (Internet IP address) with a specific private address. Static translation ensures the software always maps the same public address to a given private address. For example, you can map 10.1.1.1 to 150.1.1.1. Use static NAT when you want a specific host in the private network to always use the same Internet address when communicating outside the private network. ServerIron ADX supports both inside to outside network translation and outside to inside network Nat translation.

- **Dynamic NAT** — Maps private addresses to Internet addresses. The Internet addresses come from a pool of addresses that you configure. For example, you can dynamically translate the global pool 150.1.1.10 - 19 to private pool 10.1.1.1 - 254. In Figure 16, the pool is the range of addresses from 209.157.1.2/24 – 209.157.1.254/24. With dynamic NAT, the software uses a round robin technique to select a global IP address to map to a private address from a pool you configure.

  Dynamic NAT uses Port Address Translation (PAT). Otherwise, the return traffic cannot be reliably de-multiplexed to the correct internal client.

NOTE
You can configure both dynamic and static NAT on the same device. When you configure both types of NAT, static NAT takes precedence over dynamic NAT. Thus, if you configure a static NAT translation for a private address, the ServerIron ADX always uses that translation instead of creating a dynamic one.

Configuring static NAT

Use the **ip nat inside source static** command to explicitly map a private address to an Internet address. Static NAT ensures a specific host in the private network is always mapped to the Internet address you specify.

To map a private address 10.10.10.69 to an Internet address 209.157.1.69, enter the command such as the following.

```
ServerIronADX(config)# ip nat inside source static 10.10.10.69 209.157.1.69
```

Syntax: [no] ip nat inside source static <*private-ip*> <*global-ip*> [<*priority*>] **list** [<*acl-id*>]

The <*private-ip*> variable specifies the private IP address.

The <*global-ip*> variable specifies the IP address. The ServerIron ADX supports up to 255 global IP addresses.

The <*priority*> variable specifies a value of 1 or 2 and enables static NAT redundancy. A value of 2 means higher priority, and will be the owner of the NAT IP as long as the system is up.

The **list** parameter specifies the access list identified by the <*acl-id*> variable that will permit only the configured tcp or udp port numbers.

Configuring dynamic NAT

To configure dynamic NAT, perform the following tasks:

- Configure a standard or extended ACL for each private address range for which you want to provide NAT.

  NOTE
  Named ACLS are not supported with NAT. You must use a numbered ACL.

- Configure a pool for each consecutive range of Internet addresses to which you want NAT to be able to map the private addresses specified in the ACLs. Each pool must contain a range with no gaps. If your Internet address space has gaps, configure separate pools for each consecutive range within the address space.
- Associate a range of private addresses (specified in a standard or extended ACL) with a pool.

Configuring an address pool

Use the **ip nat pool** command to configure the address pool. For an example, refer to "Dynamic NAT configuration example 1" on page 96.

Syntax: **[no] ip nat pool** <*pool-name*> <*start-ip*> <*end-ip*> **netmask** <*ip-mask*> **| prefix-length** <*length*> **| port-pool-range** <*priority-value*>

The <*pool-name*> parameter specifies the name assigned to the pool. It can be up to 255 characters long and can contain special characters and internal blanks. If you use internal blanks, you must use quotation marks around the entire name.

The <*start-ip*> parameter specifies the IP address at the beginning of the pool range. Specify the lowest-numbered IP address in the range.

The <*end-ip*> parameter specifies the IP address at the end of the pool range. Specify the highest-numbered IP address in the range.

NOTE
The address range cannot contain any gaps. Make sure you own all the IP addresses in the range. If the range contains gaps, you must create separate pools containing only the addresses you own.

The **netmask** <*ip-mask*> | prefix-length <*length*> parameter specifies a classical sub-net mask (example: netmask 255.255.255.0) or the length of a CIDR prefix (example: prefix-length 24). The ServerIron ADX supports up to 255 global IP addresses.

The **port-pool-range** <*priority-value*> parameter enables dynamic NAT redundancy, where the <*priority-value*> can be 1 or 2. A range value of 2 indicates higher priority for the NAT IP. A 2 value also means the source ports allocated for the NAT IP are from the higher range.

Associating a range of private addresses with a pool and enabling PAT

Use **ip nat inside source list** to associate a private address range with a pool of Internet addresses and enable PAT. For an example, refer to "Dynamic NAT configuration example 1" on page 96.

Syntax: **[no] ip nat inside source list** <*acl-id*> **pool** <*pool-name*>

The **inside source** keyword specifies that the translation applies to private addresses sending traffic to the Internet (inside source).

The **list** <*acl-id*> parameter specifies a standard or extended ACL. Named ACLS are not supported with NAT. You must use a numbered ACL.

The **pool** <*pool-name*> parameter specifies the pool name. You must create the pool before you can use it with this command.

NAT configuration examples

The following sections provide both Dynamic and Static NAT configuration examples.

NOTE
A ServerIron ADX can have a maximum of 255 global IP addresses, in a single pool or multiple pools.

Dynamic NAT configuration example 1

This section describes the Dynamic NAT configuration shown in Figure 16.

FIGURE 16 Minimum required commands

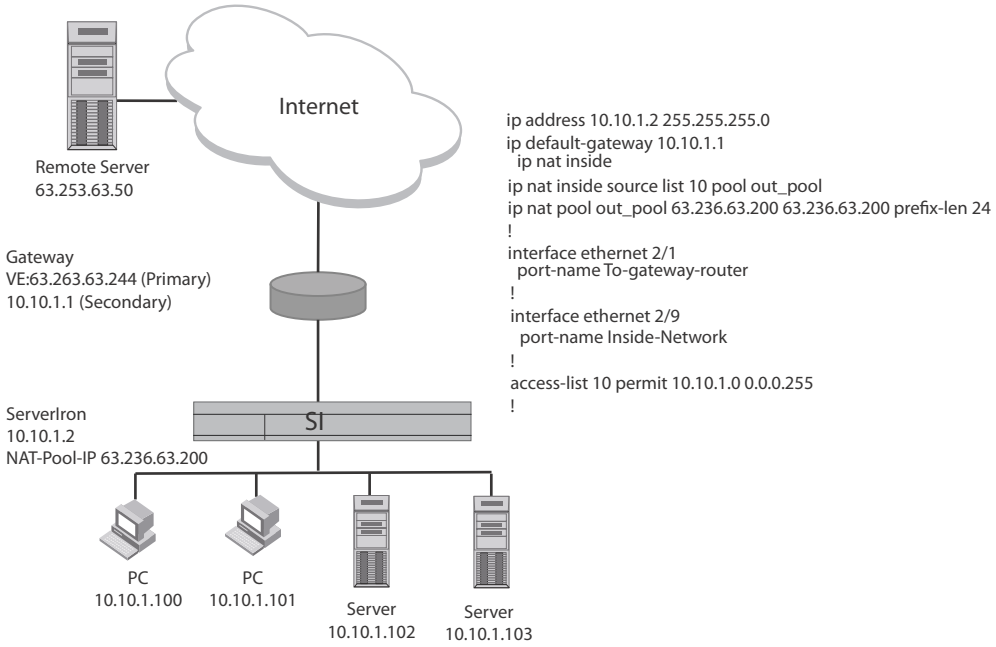


Figure 16 shows an example of a network using dynamic NAT on a ServerIron ADX ADX. The device is acting as a gateway to connect a private network to the Internet. The private network, which can also be considered as the **inside** network, is using IP addresses in the range of 10.10.1.2 - 10.10.1.254 with a 24-bit subnet mask.

The ServerIron ADX is connected to the Internet through a router. The **outside** interface of the ServerIron ADX has a global IP address of 209.157.1.1. The ServerIron ADX also has a pool of global IP addresses, which are used to map internal IP addresses.

Minimum required commands for dynamic NAT configuration.

1. Identify an internal and external interface on the ServerIron ADX. In this example, Ethernet 1/2 and 1/1 are used.

```
int eth 1/2
int eth 1/1
```

2. Assign IP addresses to the interfaces and define the outside and inside boundaries of the NAT mechanism.

```
ServerIronADX(config)# int eth 1/2
ServerIronADX(config-if-1/2)# ip address 209.157.1.1/24
ServerIronADX(config-if-1/2)# ip nat outside
ServerIronADX(config-if-1/2)# int eth 1/1
ServerIronADX(config-if-1/1)# ip address 10.10.10.1/24
ServerIronADX(config-if-1/1)# ip nat inside
```

On Switch (S) code, enable NAT globally.

```
ServerIronADX(config)# ip nat inside
```

On Router (R) code, enable NAT on interfaces (both **ip nat inside** and **outside** should be enabled). The interfaces can also be physical interfaces (not necessarily virtual interfaces).

```
ServerIronADX(config-ve-2)#ip nat inside
ServerIronADX(config-ve-3)#ip nat outside
```

3. Configure a numbered ACL and permit the IP addresses on the inside. Then define the global address pool and enable dynamic NAT.

```
ServerIronADX(config)# access-list 101 permit ip 10.10.1.0/24 any
ServerIronADX(config)# ip nat pool global_pool 209.157.1.2 209.157.1.254
prefix-length 24
```

Make sure you specify **permit** in the ACL, rather than **deny**. If you specify **deny**, the ServerIron ADX will not provide NAT for the addresses.

4. Tie the inside source list to the global pool and enable PAT (overload) to send traffic out the external interface.

```
ServerIronADX(config)# ip nat inside source list 101 pool global_pool
```

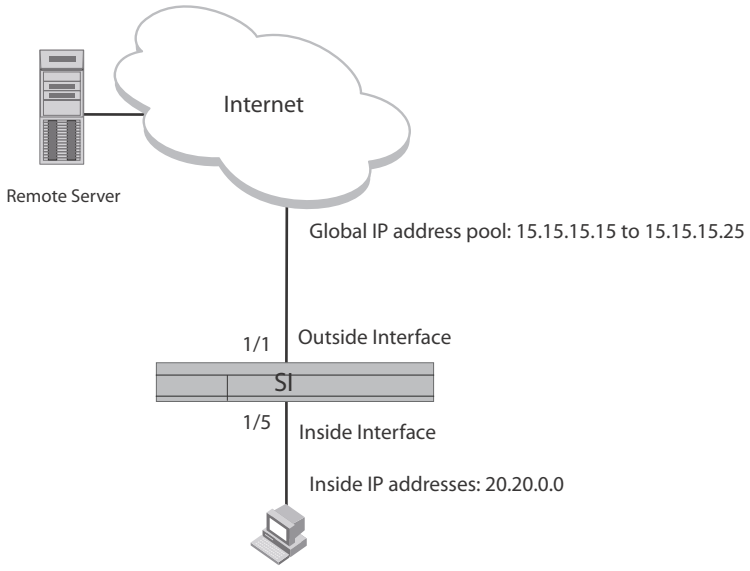
5. **rconsole** into the BP and verify the translation is working correctly.

```
rconsole x/x
show ip nat statistic
show ip nat translation
```

Dynamic NAT configuration example 2

In the following example, the ServerIron ADX is configured to translate inside hosts in the 20.20.0.0 network to unique global addresses in the 15.15.15.15/24 network.

FIGURE 17 Example of a dynamic NAT configuration - translating inside host addresses to unique pool addresses



This example requires that Interfaces 1/5 and 1/1 be configured as Inside and Outside interfaces respectively as shown.

```
ServerIronADX(config)# interface ethernet 1/5
ServerIronADX(config-if-e1000-1/5) ip address 20.20.50.1 255.255.0.0
ServerIronADX(config-if-e1000-1/5) ip nat inside
ServerIronADX(config)# interface ethernet 1/1
ServerIronADX(config-if-e1000-1/5) ip address 30.30.0.1 255.255.0.0
ServerIronADX(config-if-e1000-1/5) ip nat outside
```

The following command creates a pool of IP NAT addresses from 15.15.15.15 to 15.15.15.25 named p1.

```
ServerIronADX(config)# ip nat pool p1 15.15.15.15 15.15.15.25 prefix-len 24
```

An ACL is created to permit traffic from inside hosts in the 20.20.0.0 network as shown.

```
ServerIronADX(config)# access-list 1 permit 20.20.0.0 0.0.255.255
```

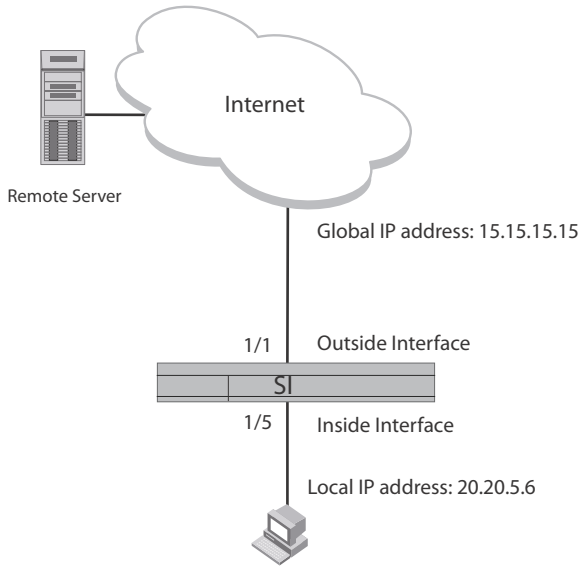
The following command ties the inside source list defined in ACL “1” to the pool named “p1” and enables PAT to send traffic out the interface defined as “outside”.

```
ServerIronADX(config)# ip nat inside source list 1 pool p1
```

Static NAT configuration example

The following examples describe how to configure a Static NAT configuration for Inside to Outside and Outside to Inside translation for the example shown in Figure 18.

FIGURE 18 Example of a static NAT configuration using router code



Configured for inside to outside translation

In the following example, the ServerIron ADX is configured to translate the local host IP address 20.20.5.6 to the unique global address 15.15.15.15.

This example requires that Interfaces 1/5 and 1/1 be configured as Inside and Outside interfaces respectively as shown.

```
ServerIronADX(config)# interface ethernet 1/5
ServerIronADX(config-if-e1000-1/5) ip address 20.20.50.1 255.255.0.0
ServerIronADX(config-if-e1000-1/5) ip nat inside
ServerIronADX(config)# interface ethernet 1/1
ServerIronADX(config-if-e1000-1/5) ip address 30.30.0.1 255.255.0.0
ServerIronADX(config-if-e1000-1/5) ip nat outside
```

The following command configures the ServerIron ADX to translate IP packets with a local IP address of 20.20.5.6 to the global IP address 15.15.15.15.

```
ServerIronADX(config)# ip nat inside source static 20.20.5.6 15.15.15.15
```

Configured for outside to inside translation

To configure the network shown in Figure 18 for Outside to Inside translation the only requirement is that the Interface configured as an Outside interface must be configured with an additional IP address in the 15.15.15.0/24 network as shown in the following.

```
ServerIronADX(config)# interface ethernet 1/1
ServerIronADX(config-if-e1000-1/5) ip address 30.30.0.1 255.255.0.0
ServerIronADX(config-if-e1000-1/5) ip address 15.15.15.100 255.255.0.0
ServerIronADX(config-if-e1000-1/5) ip nat outside
```

PAT

Dynamic NAT uses Port Address Translation (PAT). Since there is no one-to-one mapping between private addresses and global addresses, PAT maps a client's IP address and TCP/UDP port to both a global IP address and a TCP/UDP port. In this way, the ServerIron ADX can map many private addresses to the same public address and use TCP/UDP ports to uniquely identify the private hosts.

PAT maps a client’s IP address and TCP or UDP port number to both an IP address and a TCP or UDP port number. In this way, the ServerIron ADX can map many private addresses to the same public address and use TCP or UDP port numbers to uniquely identify the private hosts.

NOTE
PAT is also called **overloading** an inside global address.

Example

Inside address	Outside address
10.10.10.2:6000	209.157.1.2:1024
10.10.10.3:6000	209.157.1.2:1025
10.10.10.4:6000	209.157.1.2:1026

NAT is mapping the same global IP address to three different private addresses along with their TCP or UDP ports, but uses a different TCP or UDP port number for each private address to distinguish them. Notice that the PAT feature does not attempt to use the same TCP or UDP port number as in the client’s packet.

Forwarding packets without NAT translation

When ServerIron ADX receives a non-SYN packet for a TCP flow from an internal NAT client and no sessions are found, then by default ServerIron drops that packet. Optionally, you can forward that packets without NAT translation by entering the following command.

```
ServerIronADX(config)# nat-forward-no-session
```

Syntax: [no] nat-forward-no-session

Translation timeouts

The NAT translation table contains all the currently active NAT translation entries on the device. An active entry is one the ServerIron ADX creates for a private address when the client at that address sends traffic.

NAT performs the following steps to provide an address translation for a source IP address:

- NAT looks in the translation table for an active NAT entry for the translation. If the table contains an active entry for the session, the ServerIron ADX uses that entry.
- If NAT does not find an active entry in the NAT translation table, NAT creates an entry and places the entry in the table. The entry remains in the table until the entry times out.

Each NAT entry remains in the translation table until the entry ages out.

Configuring the NAT translation aging timer

Use the **ip nat translation** command to alter the NAT translation aging timer.

The NAT translation table contains all the currently active NAT translation entries on the device. An active entry is one that the ServerIron ADX created for a private address when that client at that address sent traffic to the Internet. NAT performs the following steps to provide an address translation for a source IP address:

- The feature looks in the NAT translation table for an active NAT entry for the translation. If the table contains an active entry for the session, the ServerIron ADX uses that entry.
- If NAT does not find an active entry in the NAT translation table, NAT creates an entry and places the entry in the table. The entry remains in the table until the entry times out.

```
ServerIronADX(config)# ip nat translation tcp-timeout 1800
```

Syntax: [no] ip nat translation dns-timeout | finrst-timeout | icmp-timeout | syn-timeout | tcp-timeout | udp-timeout <*secs*> **maximum**

Syntax: [no] ip nat translation max-entries <*number-of-entries*>

The **dns-timeout** keyword indicates connections to a Domain Name Server (DNS). The default is 120 seconds.

The **finrst-timeout** keyword identifies TCP FIN (finish) and RST (reset) packets, which normally terminate TCP connections. The default is 120 seconds. This timer is not related to **tcp-timeout**, which applies to packets to or from a host address that is mapped to an global IP address and a TCP port number (PAT feature). The **finrst-timeout** applies to packets that terminate a TCP session, regardless of the host address or whether PAT is used.

The **icmp-timeout** keyword indicates timeout for NAT ICMP flows

The **syn-timeout** keyword indicates timeout for NAT TCP flows after a SYN

The **tcp-timeout** keyword indicates dynamic entries that use PAT based on TCP port numbers. The default is 120 seconds. This timer applies only to TCP sessions that do not end “gracefully”, with a TCP FIN or TCP RST.

The **udp-timeout** keyword indicates dynamic entries that use PAT based on UDP port numbers. The default is 120 seconds.

The <*secs*> parameter specifies number of seconds, 0– 3600. Use **maximum** to set the maximum timeout value. For example, 3,600 seconds.

The **max-entries** <*number-of-entries*> parameter specifies the maximum number of NAT entries

Stateless static IP NAT

A ServerIron ADX creates sessions for Static NAT by default. You can prevent a ServerIron ADX from creating sessions for static NAT traffic with the following command.

```
ServerIronADX(config)# [no] ip nat stateless
```

Syntax: ip nat stateless

For “ip nat stateless“ to work, the existing command, “ip nat inside source static” must already be configured.

Example

```
ip nat inside source static 10.45.16.103 10.45.16.10
```

NOTE
FTP, RTSP and other similar complex protocols are not supported. The traffic applicable for IP NAT Stateless are TCP, UDP, and ICMP.

NOTE
You must reload a ServerIron ADX whenever changes are made to a running IP NAT configuration.

Redundancy

The IP NAT Redundancy feature implements a separate protocol to negotiate IP address ownership of NAT IP addresses.

The new protocol is similar to the symmetric VIP protocol and uses any L2 link to exchange the NAT PDUs. Both ServerIronADXs will run a “symmetric VIP like” protocol to report and receive ownership (similar to the VLAN AD protocol in symmetric SLB). When one ServerIron ADX goes down, the peer ServerIron ADX will become the master for that NAT IP (in case of static NAT) or NAT pool (in case of dynamic NAT). However, the NAT IP/NAT pool ownership is used only to decide which ServerIronADX responds to the ARP request for the NAT IP. Both ServerIronADXs are allowed to use the NAT IP in keeping with the design for symmetric VIP (sym-active SLB).

The global **ip policy** dependency is as follows:

- SLB — not needed
- IP NAT — not needed

- TCS — An **ip policy** must be defined. Without it, caching will not work.

Enabling IP NAT

When a ServerIron ADX is configured with Switch code, NAT is enabled globally but when it is configured with Router Code, it is enabled per-interface.

NOTE
ServerIron ADX ADX does not support IP NAT inside and outside on the same physical interface.

Enabling IP NAT globally

The following command enables IP NAT globally.

```
ServerIronADX(config)# ip nat inside
```

Syntax: [no] ip nat inside

Enabling IP NAT per-interface

When enabled per-interface, IP NAT must be enabled exclusively “inside” or “outside” on a physical or virtual interface as shown in the following example.

```
ServerIronADX(config)# interface ethernet 1/5
ServerIronADX(config-if-e1000-1/5) ip nat inside
```

Syntax: [no] ip nat [inside | outside]

The **inside** parameter configures the interface as an IP NAT inside interface.

The **outside** parameter configures the interface as an IP NAT outside interface.

Enabling static NAT redundancy

To enable static NAT redundancy, enter the **ip nat inside source static** command such as the following.

```
ServerIronADX(config)# ip nat inside source static 10.10.10.10 63.32.23.1 2
```

Syntax: ip nat inside source static <*ip-addr1*> <*ip-addr2*> <*priority-value*>

The existing **ip nat inside** command has been extended to include a <*priority-value*>, which is used to determine the owner of the NAT IP address.

The <*priority-value*> can be 1 or 2. 2 is the higher priority, and will be the owner of the NAT IP as long as the system is up.

Enabling dynamic NAT redundancy

To enable dynamic NAT redundancy, enter commands such as the following.

```
ServerIronADX(config)# ip nat pool foo 63.23.1.2 63.23.1.4 prefix 24
ServerIronADX(config)# ip nat pool foo port-pool-range 2
```

Syntax: ip nat pool <*pool-name*> **port-pool-range** <*priority-value*>

The **port-pool-range** <*priority-value*> parameter supports redundancy for IP NAT pool addresses. This parameter is similar to the priority value for static NAT, except it also determines the range of source ports allocated by the NAT IP (which prevents source port collision).

In ServerIron ADX, the **ip nat pool** <name> **port-pool-range** command is mandatory for running router code in HA setups. This command decides the ownership of the IP NAT pool and, when using router code, this command has to be used in tandem with **ip nat pool** <name> <Start-IP-address> <End-IP-address> command.

The <*priority-value*> can be 1 or 2. A range value of 2 indicates higher priority for the NAT IP. It also means the source ports allocated for the NAT IP are from the higher range.

NOTE
A distribution of port ranges is not required for static NAT, as it does not involve PAT.

Displaying NAT information

The following sections describe how to display NAT information.

Displaying NAT statistics

To display NAT statistics, enter commands such as the following.

```
ServerIronADX# rconsole 1 1
ServerIronADX1/1#ServerIronADX_Lower1/1# show ip nat stat
Debug counters:
 TCP FWD:
send nat unreachable tcp fwd) =0 nat tcp no ports avl = 2867811
nat tcp status zero = 0 nat tcp ip status zero = 0
nat tcp usr index null = 0
 TCP REV:
send nat unreachable (tcp rev) =0 nat tcp rev no ports avl = 0
nat tcp rev status zero = 0 nat tcp rev ip status zero = 0
nat tcp rev usr index null = 0
 UDP FWD:
send nat unreachable (udp_fwd) =0 nat udp fwd no ports avl = 0
nat udp fwd status zero = 0 nat udp fwd ip status zero = 0
nat udp fwd usr index null = 0
 UDP REV:
send nat unreachable (udp rev) =0 nat udp rev no ports avl = 0
nat udp rev status zero = 0 nat udp rev ip status zero = 0
nat udp rev usr index null = 0
nat corruption = 0 rtsp port unavailable = 0
RTSP inside alloc same = 53271 RTSP reply port not same= 0
Wrong port range = 0
Total translations: 12433 (0 static, 12433 dynamic)
Hits: 5786108 Misses: 94
Expired translations: 5773769
Dynamic mappings:
 pool p1: prefix_len= 24
     start 15.15.15.15 end 15.15.15.20
     total addresses 6 overloaded
[0]: h: 0 t: 0 m: 248a6000 T: 384 f: 384
[1]: h: 0 t: 0 m: 248ab000 T: 384 f: 384
[2]: h: 0 t: 0 m: 29f33000 T: 384 f: 384
[3]: h: 0 t: 0 m: 2a5cb000 T: 384 f: 384
[4]: h: 0 t: 0 m: 2a5d4000 T: 384 f: 384
[5]: h: 0 t: 0 m: 2a5dd000 T: 384 f: 384
[0]: h: 0 t: 0 m: 248a8000 T: 320 f: 320
[1]: h: 0 t: 0 m: 29f30000 T: 320 f: 320
[2]: h: 0 t: 0 m: 2a5c4000 T: 320 f: 320
[3]: h: 0 t: 0 m: 2a5cd000 T: 320 f: 320
[4]: h: 0 t: 0 m: 2a5d6000 T: 320 f: 320
[5]: h: 0 t: 0 m: 2a5df000 T: 320 f: 320
 [0]: h: 972 t: 5653 m: 248a2000 T: 7168 f: 4681
 [1]: h: 972 t: 5653 m: 2a5bc000 T: 7168 f: 4681
 [2]: h: 2520 t: 656 m: 2a5c0000 T: 7168 f: 5304
 [3]: h: 2520 t: 655 m: 2a5c7000 T: 7168 f: 5303

Sess: Total 2000000, Avail 1950226, NAT 24886
Stream media=1, RTSP=(1:94547), MMS=(1:0), PNM=(1:0)
Inside global      Last Inside Local  xmit pkts  xmit bytes rx pkts    rx byt
15.15.15.15        20.20.2.15         89909      7619664    44954      80235
15.15.15.16        20.20.2.16         89907      7619447    44950      80235
15.15.15.17        20.20.2.11         67428      5714424    33712      60175
15.15.15.18        20.20.2.12         67428      5714424    33712      60175
15.15.15.19        20.20.2.13         67432      5714763    33714      60179
15.15.15.20        20.20.2.14         67434      5714883    33714      6017
```

Syntax: show ip nat statistics

TABLE 14 Display fields for show ip nat statistics

This field...	Displays...
send nat unreachable (tcp fwd)	Indicates the number of times that a “port unreachable” message was generated for NAT TCP forward traffic.
nat tcp no ports avl	Indicates the number of times that a “port unreachable” message was generated because the ServerIron could not get a port from the port pool for a NAT IP for TCP forward traffic.
nat tcp status zero	Indicates the number of times that an error in NAT translation for TCP forward traffic.
nat tcp ip status zero	Indicates errors in NAT translation for TCP forward traffic.
nat tcp usr index null	Indicates the number of times that a “port unreachable” message was generated because the ServerIron could not create a user session for TCP forward traffic.
send nat unreachable (tcp rev)	Indicates the number of times that a “port unreachable” message was generated for TCP reverse traffic.
nat tcp rev no ports avl	Indicates the number of times that a “port unreachable” message was generated because the ServerIron could not get a port from the port pool for an IP NAT for TCP reverse traffic.
nat tcp rev status zero	Indicates the number of times that an error in NAT translation for TCP reverse traffic has occurred.
nat tcp rev ip status zero	Indicates the number of times that an error in NAT translation for TCP reverse traffic has occurred.
nat tcp rev usr index null	Indicates the number of times that a “port unreachable” message was generated because the ServerIron could not create a user session for TCP reverse traffic.
send nat unreachable (udp fwd)	Indicates the number of times that a “port unreachable” message was generated for UDP forward traffic.
nat udp fwd no ports avl	Indicates the number of times that a “port unreachable” message was generated because the ServerIron could not get a port from the port pool for a NAT IP for UD forward traffic.
nat udp fwd status zero	Indicates the number of times that an error in NAT translation for UDP forward traffic has occurred.
nat udp fwd status zero	Indicates the number of times that an error in NAT translation for UDP forward traffic has occurred.
nat udp fwd usr index null	Indicates the number of times that a “port unreachable” message was generated because the ServerIron could not create a user session for UDP forward traffic.
send nat unreachable (udp rev)	Indicates the number of times that a “port unreachable” message was generated for UDP reverse traffic because the Serveriron could not get a port from the port pool for a NAT IP.
nat udp rev no port avl	Indicates the number of times that a “port unreachable” message was generated because the ServerIron could not get a port from the port pool for a NAT IP for UDP reverse traffic.
nat udp rev status zero	Indicates the number of times that an error in NAT translation for UDP reverse traffic has occurred.

TABLE 14 Display fields for show ip nat statistics (Continued)

This field...	Displays...
nat udp rev ip status zero	Indicates the number of times that an error in NAT translation for UDP reverse traffic has occurred.
nat udp rev usr index null	Indicates the number of times that a “port unreachable” message was generated because the ServerIron could not create a a user session for UDP reverse traffic.
sw l4 nat corruption	Indicates the number of instances of NAT session corruption.
rstp port unavailable	Indicates the number of times that a NAT port was not available for RSTP.
RTSP inside alloc same	Indicates the number of times that the used port and proposed client port were the same for RSTP.
RTSP reply port not same	Indicates the number of times that the used port and proposed client port were not the same for RTSP.
Wrong port range	Indicates the number of times that the NAT port used a port in the wrong port range. For example, where a NAT port used a port from the normal port pool range for RTSP.
Port Pool Parameters	
[x]	The variable represented by "x" represents the index of the IP address in the IP NAT pool. For example, [0] refers to the first IP address in the IP pool (216:220:209:230). [1] refers to the second IP address in this IP pool (216:220:209:231).
h	The value following "h:" refers to the head of the port pool for the IP address in the IP NAT pool. The head indicates the location in the port pool where the next port will be allocated from.
t	The value following "t:" refers to the tail of the port pool for the IP address in the IP NAT pool. The tail indicates the location in the port pool where the next port will be freed from.
T	The value following "T:" refers to the total number of ports in the port pool for that IP address in the IP NAT pool.
f	The value following "f:" refers to the number of free ports in the port pool for this IP address.

Displaying NAT translation

To display the currently active NAT translations, enter the following command.

```
ServerIronADX# show ip nat translation

Pro Inside global       Inside local       Outside local    Outside global
tcp 10.1.1.92:11021       5.1.1.2:32784        10.1.1.1:23       10.1.1.1:23
```

Syntax: show ip nat translation

NOTE
You can enter this command only when you **rconsole** in to a BP. The command is not supported on the Main Processor CPU.

TABLE 15 Display fields for show ip nat translation

This field...	Displays...
Pro	When PAT is enabled, this field indicates the protocol NAT is using to uniquely identify the host. NAT can map the same IP address to multiple hosts and use the protocol port to distinguish among the hosts. This field can have one of the following values: • **tcp** – In addition to this IP address, NAT is associating a TCP port with the host on the private network. • **udp** – In addition to this IP address, NAT is associating a UDP port with the host on the private network.
Inside global	The Internet address mapped to the private address listed in the Inside local field for inside NAT.
Inside local	The private address mapped to the Internet private address listed in the Inside global field for inside NAT.
Outside global	The destination of the traffic. If PAT is enabled, the TCP or UDP port also is shown. **NOTE:** Outside NAT is not supported.
Outside local	The destination of the traffic. If PAT is enabled, the TCP or UDP port also is shown. **NOTE:** Outside NAT is not supported.

Displaying NAT redundancy information

You can display information about the state of the static NAT IP or NAT pool (dynamic), the MAC address used, and the configured priority. The MAC address used for the NAT IP is a special construct, where the last 3 bytes of the MAC address are derived from the shared NAT IP address (similar to the symmetric MAC).

To display NAT redundancy information, enter the following command.

```
ServerIronADX# show ip nat redundancy (on active)
NAT Pool Start IP: 10.1.1.150 Mac address: 020c.db01.0196
State: Active Priority: High
NAT Pool Start IP: 10.1.1.91   Mac address: 020c.db01.015b
State: Active Priority: High
NAT Pool Start IP: 10.1.1.92   Mac address: 020c.db01.015c
State: Active Priority: High
NAT Pool Start IP: 10.1.1.95   Mac address: 020c.db01.015f
State: Active Priority: High
```

The two “**Priority**” options are “**High**” and “**Low**”. That is, 2 or 1.

The two “**State**” options are “**Active**” and “**standby**”.

```
ServerIronADX# show ip nat redundancy (on standby)
NAT Pool Start IP: 10.1.1.150 Mac address: 020c.db01.0196
State: Standby Priority: Low
Standby Idle count: 0 Threshold: 20
NAT Pool Start IP: 10.1.1.91   Mac address: 020c.db01.015b
State: Standby Priority: Low
Standby Idle count: 0 Threshold: 20
NAT Pool Start IP: 10.1.1.92   Mac address: 020c.db01.015c
State: Standby Priority: Low
Standby Idle count: 0 Threshold: 20
NAT Pool Start IP: 10.1.1.95   Mac address: 020c.db01.015f
State: Standby Priority: Low
Standby Idle count: 0 Threshold: 20
```

Syntax: show ip nat redundancy

Displaying VRRPE information

To display VRRPE information, enter the following command.

```
ServerIronADX_Lower# show ip vrrp-e brief

Total number of VRRP-Extended routers defined: 2
Interface VRID CurPri P State  Master addr   Backup addr       VIP

 v5         1    125 P Master Local         Unknown         5.1.1.9
 v10        2    125 P Master Local         Unknown         10.1.1.9
```

Syntax: show ip vrrp-e brief

Clearing NAT entries from the table

Use the **clear ip nat** command to manually clear entries from the NAT table.

Syntax: clear ip nat <*protocol*> **inside** <*global-ip*> <*global-port*> <*private-ip*> <*local-port*>

The <*protocol*> parameter specifies the protocol type and can be **tcp** or **udp** plus its global or local port number.

To clear a specific NAT entry based on the private and global IP addresses, enter the command such as the following.

```
ServerIronADX# clear ip nat inside 209.157.1.43 10.10.10.5
```

This command clears the inside NAT entry that maps private address 10.10.10.5 to Internet address 209.157.1.43.

Syntax: clear ip nat inside <*global-ip*> <*private-ip*>

To clear all static and dynamic entries from the NAT translation table, enter the following command.

```
ServerIronADX# clear ip nat all
```

Syntax: clear ip nat all